THE UNIVERSITY
of EDINBURGH

Compositional Taylor Model Based Validated Integration

Kristjan Liiva

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2019

Abstract

Validated integration is a family of methods that compute enclosures for sets of initial conditions in the Initial Value Problems. The Taylor model based validated integration methods use truncated Taylor series to approximate the solution to the Initial Value Problem and often give better results than other validated integration methods. Validated integration methods, and especially Taylor model based ones, become increasingly more impractical as the number of variables in the system get higher.

In this thesis, we develop techniques that mitigate the issues related to the dimension of the system in Taylor model based validated integration methods. This is done by taking advantage of the compositional structure of the problem when possible. More precisely, the main contribution of this thesis is to enable computing an enclosure to a higher dimensional system by using enclosures for smaller lower dimensional subsystem that are contained in the larger system.

The techniques called shrink wrapping and preconditioning are used in the Taylor model based validated integration to improve accuracy. We also analyse these techniques from a compositional viewpoint and present their compositional counterparts.

We accompany compositional version of the Taylor model based validated integration with implementation of our tool CFlow* and experiments using our tool. The experimental results show performance gains for some systems with non-trivial compositional structure.

This work was motivated by interest in formally analysing biological systems and we use biological systems examples in a number of our systems.

Acknowledgements

I would like to thank the people who have been important to my PhD studies. This work would have not been possible without their support and guidance.

First and foremost, I would like to express my sincerest gratitude to my PhD supervisor, Paul B. Jackson. Paul's door was ever open and he provided me with the best guidance, support, positivity and encouragement I could have hoped for. His expertise in how to continue when facing technical difficulties was invaluable. I know that at times I did not make supervising easy, but I am very grateful that it was exactly Paul who was my supervisor, I cannot imagine getting better help than what I got from Paul from anybody else.

I would also like to thank my second supervisor, Grant O. Passmore. Grant's thesis was the main motivator for the original direction of my studies and his input in its evolution was as impactful as Paul's. Although during my studies Grant became heavily involved with his (very successful) aspirations in the private sector, he was still very present and continuously helpful in my studies. Much more than I could have expected. I am also very grateful for the short abstract discussion we held at times, Grant's enthusiasm and warmth for the appreciation of mathematical beauty were infectious and invigorating.

My sincere thanks also extend to my supervisor from industry, Christoph M. Wintersteiger. He provided with very useful alternative insight into problems that I or my other supervisor often could not see. I am also very grateful for his ever helpful presence during my visits to Microsoft Research Cambridge.

I would like to thank all 3 of my supervisors together for being available for regular discussions and meetings. At the times of discouragement, I never failed to feel much more positive after these meetings. Besides each of them being irreplaceable in the making of this thesis, their involvement also greatly improved my doctoral experience.

I would also like to thank Jacques Fleuriot and Jane Hillston for agreeing to be on my yearly review meeting panels, and for all the feedback and constructive criticism they provided.

I am also grateful to Ian Stark and Lawrence C. Paulson for agreeing to be my examiners, for spending many hours of their time carefully reading through a thesis, and for all their feedback that helped to greatly improve the final version of this thesis.

Much of my work would not have been possible without Erika Ábrahám and Xin Chen, their tool Flow* and their helpful support when discussing it.

I would like to thank George Corliss for the lengthy warm, insightful and encouraging discussion we had. It helped a lot when the direction of my studies changed.

I would like to thank Paul Anderson as the lecturer of the Introduction to Java Programming course, I really enjoyed being involved in the teaching of that course and interacting with him was always a pleasure.

I would like to express my thanks to LFCS and its members for providing an excellent research environment.

I would like to thank my friends: Danel Ahman, Veselin Blagoev, Weili Fu, Lilian Liiva, Theodoros Kapourniotis, Vladimir Nikishkin, Chrystalla Pavlou and Marcin Szymczak.

From my undergraduate at the Tallinn University of Technology, I have Aivo Anier, Alar Leibak, Enno Pais, Peeter Puusemp, Tõnu Ruus, Ennu Rüstern, Ivar Tammeraid, Tanel Tammet, Tarmo Uustalu and Jüri Vain to thank. They showed me how enjoyable understanding and learning can be. I'd also like to thank Juhan Ernits for pointing me to Grant's thesis.

A big thank you goes to my family. I'm especially grateful to my mother and how she has believed in, supported and loved me.

I cannot overestimate the effect that my girlfriend Zoe Tian has had on me. Her unconditional love has helped me through my toughest times and her encouragement and support gave me motivation when I needed it. Zoe, thank you for showing me the colours again.

Finally, I would like to acknowledge the financial support of the University of Edinburgh in the first year of my studies (through EPSCRC DTA Award) and Microsoft Research for the remainder of it (through University of Edinburgh Microsoft Research Joint Initiative in Informatics). I am also grateful for the University of Edinburgh for funding me while this thesis was under examination.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Kristjan Liiva*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents an ordinary differential equation (ODE) solver. The angle used is that of the validated integration, i.e. to provide mathematically proven bounds to the solution. The main novelty is to make use of the compositional nature of the system often appearing in domains such as synthetic biology.

1.1 Motivation from Biology

Together with the rise in computing power, the areas of DNA computation (using DNA strands to construct computational circuits) and synthetic biology (building genetic networks within organisms) have also evolved explosively [QW11, PW09]. What was just a couple of decades ago in the domain of science fiction is becoming a reality. Examples that include nanoparticles working on tasks such as delivering drugs and imaging [KRO05], molecular level motors delivering cargo by navigating predefined tracks [Hes11] and novel promising ways of sustainable production of biofuels and other materials [ZRK11]. Or further understanding of biological systems through knowledge and expertise gained from building similar ones artificially.

The most relevant strategy for constructing molecular nanorobots to this thesis is DNA nanotechnology [See10, ZS11]. DNA nanotechnology has enjoyed a wealth of research in the elements required for constructing nanorobots (such as sensors and amplifiers [DP04], circuits [SSZW06, Car13, QW11], motors [MBT11, OSS09] and structures [WLWS98, Rot06]). Using these elements researchers have constructed digital logic circuits and Boolean neural networks with a complex structure solely from DNA strands [QW11, QWB11].

A practical example of a framework can be found in [CDS$^+$13] where a signalling

protocol based on short single-stranded DNA sequences is presented. The essential idea is to build a system which receives inputs in the form of DNA sequences and produces outputs in the form of other sequences. We note that DNA components are capable of producing the same class of behaviours as chemical reaction networks (CRNs) [Car13, SSW10]. There has been a significant amount of study of CRNs. We direct attention to that CRNs themselves can be viewed as a general framework for modelling systems with many interacting components, such as gene regulatory networks, animal populations and sensor networks. CRNs can embody a wide range of digital and analogue behaviours [CDS+13].

Like all engineered systems, most synthetic biology systems are designed by building larger systems from smaller components. Tools such as Visual DSD [LYCP11, PC09, Mic15a, Mic15b] make it possible to do this *in silico*. The benefit of such tools is further enhanced by the fact that it is possible to eliminate the expensive *in vitro* experiments from the evolution of the systems.

Many applications of DNA computing and synthetic biology are *safety-critical* and require robustness. This requirement is often also appearing in the engineering and study of other computational systems (e.g. hardware and software) and a popular way to handle it is to apply formal methods in order to guarantee *correctness*.

One example of this is the use of methods based on the probabilistic model checking to prove properties about the states that a strand displacement circuit traverses [LPC+12]. Unfortunately, as of this time, this approach cannot handle realistic numbers of molecules. Another example is the use of SMT-based methods to analyse pertinent problems arising in DNA computing [YWHK13] (they present an example DNA circuit computing the square root of a 4-bit input in [YWH+13]).

Our strategy, that we will follow in this thesis, is to also apply formal methods to biological (and other similar) systems. More precisely, we will try to reason about the Initial Value Problem (IVP) underlying a CRN by using validated integration method. We will try to do that by taking into account the typical compositional structure of engineered systems.

1.2 Validated Integration and Its History

Numerical solutions to Initial Value Problems (IVPs) are of paramount importance for the analysis of hybrid and continuous systems. Unfortunately, ordinary differential equations (ODEs) underlying the IVPs in practical systems rarely have analytical

solutions. A popular way to cope with this issue is to use approximations such as simulations in place of the solutions.

While simulation has many benefits it is not a perfect fit. It only provides estimates that are 'close' to solutions, sometimes with unreliable, or even exponentially growing, error bounds. In addition to that, the behaviour of a practical system depends on its initial state and uncontrollable external factors such as disturbances and noise. The initial state of a system is generally not known precisely - one might be able to know that variables used to describe the state are within some range, but not know their precise values. Therefore, in order to answer a verification question, simulation, if used, needs to be accompanied with sensitivity analysis to go from finite executions of simulation to infinite possible behaviours of the system [GP06, DM07, Don07, Mic15b].

Although simulation is the most common technique, it is not the only one. One alternative is validated integration which in some ways is more fitting to our formal setting.

Firstly, instead of operating on points, validated integration operates with sets. If the uncertainty of the initial conditions and external factors are captured completely with appropriate sets, then validated integration can compute enclosure that will hold for all practical executions of the system. That is, no additional work is needed to extrapolate from finite executions to infinite executions.

Secondly, instead of computing approximation of the solution validated integration computes enclosure of it. This makes it possible to answer some verification questions by just applying the method. For instance, given a range of initial states and description of disturbances, does the system satisfy a certain safety property? That is, if a given unsafe region does not intersect with the enclosure computed, then we can guarantee that the system will never reach that region in any practical condition.

The field of validated integration was founded by Moore in the 60s [Moo65]. Early validated integration methods simply used interval arithmetic to address the round-off errors and discretization in computations. Traditional interval-based validated integration methods enclose the solution using intervals [Eij81], polynomials [Loh95] or Taylor series [CR96, NJP01]. In most of these methods, the mean-value theorem is used in order to reduce over-approximation from interval arithmetic.

Despite this, these methods frequently suffer from excessive imprecision due to coarse over-approximation from two sources. Firstly, validated integration methods work by enclosing the flow of a set of ODEs in interval boxes in a fixed coordinate system. Often the dynamics of the system have a non-linear or a rotational component

both of which require enlargement of the interval box if all of the dynamics are to be captured. This is called the *wrapping effect*. Secondly, the relationships between variables may be lost when they are represented by intervals. This is known as the *dependency problem*.

An important step in validated integration was the parallelepiped method. The parallelepiped method attempts to minimise the wrapping effect by using a moving coordinate system [Moo65]. Unfortunately, this approach faces challenges of stability with respect to long integration times, since the rotation is determined by a matrix that may become increasingly ill-conditioned. Lohner's QR method [Loh87] is an extension of the parallelepiped method that makes use of QR decomposition to find coordinate transformations that are stable. Despite these advances in tackling the wrapping effect, both the parallelepiped and QR methods are still prone to inefficiencies rooted in the dependency problem.

To address that shortcoming, Berz and Makino introduced the Taylor model (TM) based validated integration [BM98, Mak98]. In their approach, the flow of ODEs is handled by Taylor model arithmetic, which combines both symbolic computations and interval arithmetic. The core idea of the Taylor model based integration is that the majority of the flow of the ODEs is represented by a polynomial and all round-off and discretization errors are pushed into a remainder interval. The symbolic part is free of the dependency problem. While the interval remainder part still suffers from it, its effect is reduced since the width of the interval box is smaller. A downside of this basic Taylor model approach is that the remainder interval can never decrease, and thus the impact of the dependency problem grows with integration time. To mitigate this, Berz and Makino developed preconditioned Taylor model based integration [MB11]. In preconditioned methods, the solution is represented by a composition of Taylor models. The integration is only concerned with one of them and at each integration step, terms are moved between the models in order to minimise the overestimation introduced.

The rigour of validated integration comes at the cost of increased computational complexity. The work of this thesis began with the evaluation of various validated integration tools (CAPD [PV11], VSPODE [LS07], Flow* [CÁS12]) on a number of examples, many modelling chemical reactions in biological systems. While the Taylor model based tools (VSPODE and Flow*) performed better, we encountered significant problems. In many cases, the validated integration diverged well before reaching time limits of interest. When we altered integration parameters to increase accuracy and

extend the integration time, integration became very slow, taking days.

Taylor model based methods are especially sensitive to the dimensions of the system. Unfortunately, biological systems have usually high dimensionality. For example, it is not uncommon for a system to have more than 100 dimensions. In addition to high dimensionality, biological systems very often have another property called *stiffness* that makes them hard to solve. Stiffness is the property of the system where dynamics happen on different time-scales (usually these time-scales are very different - we have observed a difference by a factor of 10000). Stiff systems are hard to solve because usually the interesting behaviour is defined by the slower dynamic, while the integration accuracy is limited by the faster dynamic. As a result, you need to make very small integration steps, where very little interesting is happening.

We have noted that the ODE systems arising from chemical reactions have considerable compositional structure. The focus of this thesis is to describe how to exploit this structure in order to reduce the dimensionality of the system and improve the performance of the Taylor model based validated integration methods.

1.3 Related Works and Tools

We have given a brief history of validated integration in the previous section. In this section, we will focus on the tools of validated integration and their novelty. The theory presented in this thesis can be used both in ODE solvers and in reachability tools for hybrid systems. We will describe some of these.

To help with the presentation we will use the notation of *flowpipe* that is defined formally in Chapter 3 and that of the *trajectory* which we consider to be an approximation of the solution in simulation.

STRONG [DRJ13] is a Matlab toolbox dealing with reachability/safety verification of hybrid systems that bridges the gap between simulation and verification. For linear systems, it is able to guarantee the robustness of the trajectory for a point initial condition (and a neighbourhood around it) by solving a Lyapunov equation. When the initial set of the system is a compact set and not a point, the set might not be covered completely. If that is the case, then the aim is to cover it as much as possible. The strategy discussed is to first generate random points as initial states then use an unbiased estimator to evaluate the percentage of the covered initial set.

Breach [DM07] is a toolbox for verification and parameter synthesis of hybrid systems. It computes the enclosure of the solution by using simulation together with

sensitivity analysis. It does it by first picking finitely many sample points from the set (possibly containing an infinite number of points), finding trajectories for these points and then padding these trajectories by an appropriate amount.

The initial set of interest is usually a compact set. An important aspect of this approach is that any point in the initial is at most a fixed distance δ away from its closest sample point. Using this distance δ and the Jacobian of the system Breach quantifies how much the trajectories of the sample points need to be inflated in order to capture the solutions corresponding to any points in the initial set.

In general, the Jacobian matrix of a system is a function of the state, resulting in a need to compute it separately for each time step. However, if the system dynamics are linear time-varying then Breach is able to take into account the fact that the Jacobian of the matrix is constant. In order to find the trajectories of the sample points, Breach uses numerical solvers and although they have errors, these errors can be taken into account by inflating the enclosure further.

CAPD [PV11] library is a collection of C++ modules that includes an interval-based validated integration solver. It supports various sets to represent enclosures (various rectangular intervals, various parallelepipeds, affine sets, balls etc.) and different ways to construct the approximation of the solution (Taylor polynomial, Bernstein polynomial). It also offered some optional innovations such as having the time step be determined by an error bound.

C2E2 [FQM+16, FKJM16] is a framework for performing bounded time verification of hybrid systems. It uses the same idea as Breach. It, however, makes some optimizations in order to be less conservative with the inflation introduced. Firstly, it uses matrix measures in order to get tighter bounds on the distance any point in the initial set can differ from the trajectory of sample points. Secondly, since it computes the Jacobian matrix symbolically it does not introduce any numerical errors on this part of the algorithm. Thirdly, it uses coordinate transformation to produce a more conservative over-approximation of the flowpipes.

C2E2 also makes use of a very basic idea of compositionality. It notes that it is problematic to deal with the Jacobian matrix when the system has some variables with constant derivatives. In order to bypass this issue, it decomposes the system automatically and handles the constant-rate part separately.

C2E2 can use the validated simulator CAPD [PV11] as well as the standard ODE solver in the Boost library to compute the trajectories of the sample points.

VNODE [JN02, NJP01] is tool implementing interval-based validated integration

method using Taylor series for enclosures. It uses Hermite-Obreschkoff method that is extended to intervals. On some problems, this leads to smaller local error, better stability and the need to use fewer operations. It is noteworthy to mention that with appropriate parameters interval Hermite-Obreschkoff method is an implicit method [Wan77]. The scheme itself is similar to other methods - a coarse enclosure is found first and then it is tightened. The tightening step, in this case, can be seen as an application of a Newton-like step. VNODE-LP [Ned11] is a successor of VNODE which is developed entirely using Literate programming. The merit of which is that the correctness of the tool can be examined easier.

[Imm14] makes the observation that although the tools used to compute enclosures are based on sound theory, there is a gap between the implementation and that theory. Namely, the proofs that the tools compute enclosures of the solution are relatively high level and they do not have a formal link to the source code.

He bridges this gap by formalising both the proof and algorithm in the interactive theorem prover Isabelle/HOL [Pau93]. This makes it possible to prove the used theorems in a detailed manner where every step is proven in rigorous calculus (modulo the existing theory of ODEs in Isabelle/HOL). Furthermore, since Isabelle/HOL implements higher-order logic which can be viewed as a functional programming language, it is possible to use Isabelle's code generator [HN10] to produce code from the formal specification (in functional programming languages like SML, OCaml, Haskell, or Scala).

The method formalized is the one presented in [BCD13] which is based on the Euler method and works with dyadic rational numbers with statically fixed precision. The discretization and round-off errors are abstracted in the domain of affine forms.

VSPODE [LS07] is a Taylor model based validated ODE solver. It employs the same techniques as discussed in [Mak98]. The novel part is that in addition to using intervals for initial values it presents a more efficient way for handling intervals for parameters.

Traditionally, time-invariant interval parameters can be treated as intervals or as additional state variables. The former case can lead to amplification of the wrapping effect and the latter case can be computationally considerably more expensive. To overcome these issues VSPODE treats parametric uncertainties directly.

In order to make bounding the terms corresponding to i^{th} order derivative in the Taylor models more tighter VSPODE also uses their Jacobian of the i^{th} order derivative to employ the mean value theorem. VPSODE uses automatic differentiation to obtain

the Taylor coefficients.

Flow* [CÁS13] performs Taylor model-based flowpipe construction for non-linear (polynomial) hybrid systems. The continuous dynamics is handled by the Taylor model based validated integration. It supports a wide variety of different options such as fixed or variable step size, fixed of variable and the heuristic selection of template directions for aggregating flowpipes. We will discuss Flow* in greater detail in Chapter 6.

KeYMaera [PQ08] is a hybrid verification tool for hybrid systems. It uses a combination of automated theorem proving, real quantifier elimination, and symbolic computations in computer algebra systems. KeYmaera is implemented as a combination of the deductive theorem prover KeY [ABHS07] and computer algebra system Mathematica or Orbital.

KeYmaera generalises KeY from discrete systems to hybrid systems by adding support for the differential dynamic logic d$\mathcal{L}$ [Pla08]. It also exploits the compositional semantics of d$\mathcal{L}$, and verifies the properties of hybrid programs by proving corresponding properties of their parts in a sequent calculus[1]. The IVPs arising in formulas are handled by solving them symbolically with the computer algebra systems Mathematica or Orbital.

dReach [KGCC15] is a bounded reachability analysis tool for hybrid systems. It encodes bounded reachability problems of hybrid systems as first-order formulas over the real numbers and solves them using δ-decision procedures in the SMT solver dReal [GKC13]. This has the effect that dReach can answer the questions about whether a system is safe (some unsafe region is never reached) or whether a system is δ-unsafe (a δ-bounded perturbation of the system can reach the unsafe region).

The motivation for using δ-decision is that a wide range of the problems are undecidable in the general sense, but decidable under δ-reachability. To solve IVPs, dReach makes use of other solvers such as VNODE [JN02, NJP01]. With an Interval Constraint Propagation [VHMK97] framework, dReach can exploit interval solvers for IVP problems, for pruning intervals on variables that appear in constraints involving ODEs.

[ERNF11] present a combination of enclosure methods for ordinary differential equations (ODEs) with the iSAT solver for large Boolean combinations of arithmetic constraints. The iSAT [FHT$^+$07] solver behaves similar to a SAT solver, but instead of checking whether a theory is satisfiable given some set of atoms, it performs a search by splitting intervals. This makes it possible to indirectly rule out atoms that are

[1] Note that is the decomposition is happening on the d$\mathcal{L}$ formulas and not continuous systems.

inconsistent under that evaluation, meaning that other arithmetic constraints must be satisfied if the entire formula is to be satisfied. Using interval constraint propagation these constraints can be used to refine the intervals for variables appearing in them.

Reasoning about ODEs can be added to this by using validated integration. ODEs in this context relate values of the variables at different time point. These values variables are often represented by intervals which means that the enclosures at a later time will also be represented by intervals. In general, a smaller input intervals in validated integration yields smaller enclosures. This is analogous to what interval constraint propagation is doing, so a similar iterative interval refinement can also take place here. Any validated solver is suitable. In [FHT+07] iSAT is combined with slightly modified version of VNODE-LP [Ned11].

The modifications to VNODE-LP lie in option to use a bracketing method. In the bracketing method, they analyse the partial derivatives and when determining that none of them are changing signs during a time step they construct two dynamical systems using the lower and upper bounds of the original local initial set. The flow of these systems can be seen as the lower and upper bound of the original system. Additionally, these constructed systems have points for initial conditions and therefore produce less over-approximation. Alternatively, if the sign of at least one derivative changes during the time step, then the bracketing method cannot be used and the unmodified VNODE-LP is used.

Similar to this thesis, Chen and Sankaranarayanan [CS16] try to address the scalability issue in the Taylor model based validated integration. Their approach is to partition the variables of a non-linear continuous system into the variables of smaller subsystems. In the context of those subsystems, the variables outside of them are viewed as time-varying uncertainties (the standard Taylor model integration technique is extended to dealing with time-varying uncertain parameters in [Che15]). Therefore instead of looking at the higher dimensional original system they consider a set of lower dimensional systems with time-varying uncertainties.

The time-varying uncertainties in [CS16] are viewed as assumption intervals. The variables abstracted using assumption intervals are assumed to be within their intervals. As long as that is the case, the decomposed system is a conservative abstraction of the original system. If a variable is outside of its assumption interval then the interval is either increased or the time step is shortened. In other words, in the context of a smaller system, this scheme resembles an on-the-fly hybridisation scheme where only the outside effects (selected variables) are hybridised.

However, using time-varying uncertainties like this results in large remainder intervals. Remainder intervals are the main source of over-approximation, so this increases the problem of over-approximation greatly. To counteract this they use symbolic remainder intervals. The idea of which is to track remainder terms symbolically over multiple steps. This helps with limiting the accumulation of overestimation in flowpipe construction. Note that it is distinct from minimizing the remainder interval at each time step which is what preconditioning is doing, so this approach can be used in conjunction with preconditioning.

The method of using symbolic remainder intervals is characterised by maximum size m of steps considered. The idea is to use previous remainder intervals when representing the remainder interval for the current step. The data structures used in relating previous step remainders to the current step are extended at each time step. This is done until they reach the size m at which point the data structures will be cleared and the remainder interval in the next step will be computed non-symbolically by the standard Taylor model based validated integration method and the process of building up the data structures is started again.

How a system is decomposed in [CS16] is framed as a problem of minimizing the number of variables that need to be abstracted in order to have smaller systems that do not affect each other and have at most a predefined number of variables. This problem can be framed as an integer linear programming problem.

A connection between decomposition presented in this thesis can be made with hybridisation [DMT10]. Hybridisation analyses a non-linear system by approximating it by an another (often linear) one that is easier to analyse.

A key aspect in hybridisation is quantifying the error of the approximation. This needs to be considered when guaranteeing that all of the trajectories of the original system are enclosed in the approximate system.

The construction of the approximate system has two important aspects. First, inside some region the vector field of the system is approximated by a simpler function. This region needs to contain the current state of the system. Often the function approximating the vector field is affine (which results in piecewise affine approximated system over the whole state space), this mainly from the fact that there has been a considerable amount of research related to the verification of such systems [ABDM00, CK03, KGBM04, Gir05, GLGM06, KV07, ASB08]. The original system is analysed by using the evolution of the approximate system when the state is fully contained inside of this region and when the state leaves this region, a new region is

constructed and the process starts again.

Second, the error bound of the approximation is estimated. This depends both on the curvature of the region and the size of the region. The estimation usually inspects how closely the affine function approximates the derivatives and then bounds the overall error using that.

Much of the research in hybridisation is related to how to partition the domain of the system and how to approximate the dynamics in region in order to yield small approximation errors [ADG03, ADG07, DLGM09]. Some methods precompute the regions and some compute them on the fly. Although it is more convenient to have a larger regions, it often increases the approximation error too much [DMT10].

NLTOOLBOX [TD13] is a library of data structures and algorithms for reachability computation of non-linear dynamical systems that uses hybridisation. It focuses on readjusting the computation parameters or exploration strategies and offers means to do that using simple C++ programs. NLTOOLBOX uses two techniques to analyse system: Bernstein expansion technique (if the system is polynomial) and hybridization (general non-linear system).

Since Bernstein expansion technique can only be applied to discrete-time systems, the system requires a system-time discretisation. Furthermore, since Bernstein expansion is only valid inside the unit box, NLTOOLBOX uses two additional techniques to guarantee that: oriented box approximation and change of variables in the polynomials.

NLTOOLBOX has been integrated into SpaceEx [FLGD$^+$11], making it possible to extend the applicability of SpaceEx to non-linear hybrid systems.

SpaceEx [FLGD$^+$11] is a verification platform for hybrid systems. The hybridisation algorithm used in SpaceEx is using variable time steps. This enables to control the size of the global error without the need to recompute trajectories for the earlier time steps.

1.4 Contributions

The main contributions of this thesis are as follows.

Compositional view of the Taylor model based validated integration. In Chapter 4, we discuss the compositional view of a system. We explain how synthetic (especially synthetic biological) systems have a modular structure and how that can be used to limit the dimensions of the system needed to consider when solving a part

of the original system. More precisely, we explain how we can define the compositional Picard operator and compositional Picard iteration focused on only supporting the parameters actually appearing in the flow for a specific variables.

Adapting the Taylor model based validated integration method to the compositional setting. Using the compositional variants of the Picard operator and Picard iteration we adapt the naive Taylor model based validated integration method to the compositional setting. To be more precise, we specify the preconditioning methods where both the left and right model are invariant w.r.t. parameter dependency. We accompany this with algorithms which makes use of this in both models or just in the left one. We adapt the method with shrink wrapping partially to the compositional setting. We explain why shrink wrapping cannot fully use the modular structure present in the compositional setting and also discuss why this does not affect the integration phase in the method with shrink wrapping.

Implementation of the tool CFlow* with experiments. We implemented a tool capable of solving systems in the compositional setting. The tool contains implementations for

- compositional naive Taylor model based validated integration,
- partially compositional Taylor model based validated integration with shrink wrapping,
- partially compositional Taylor model based validated integration with preconditioning,
- fully compositional Taylor model based validated integration with preconditioning.

CFlow* is based on the Flow* [CÁS12]. A more detailed description of our tool is given in Chapter 6.

Our tool is accompanied with an experimentation framework which we used to test our compositional version of the algorithms. We present the results of those experiments in Chapter 7.

1.5 Outline

The structure of the thesis is as follows. In Chapter 2, we present preliminaries. In Chapter 3, we will give a description of validated integration methods. In Chapter 4,

we describe the properties of systems suitable for the compositional approach, adapt validated integration methods to the compositional setting and examine the interesting cases of the preconditioned compositional approach and compositional shrink wrapping in more detail. In Chapter 6, we introduce the tool CFlow* implementing compositional validated integration. In Chapter 7, we provide our experimental results together with analysis. We present the conclusion and possible directions where this work can be taken forward in Chapter 8. Chapter 8 is followed by appendix, bibliography and nomenclature.

Chapter 2

Mathematical Preliminaries

Most of the material in this chapter will be referenced from [Che15], [Mak98] and [LS07].

2.1 Interval Arithmetic

Computers operate on floating-point numbers instead of real numbers. The reason for this is that the floating-point numbers are easier to represent and more efficient to operate with the architecture prevalently used in computers. However, most real numbers cannot be represented by floating-point numbers and the arithmetic on floating-point numbers is imprecise. Therefore, care needs to be taken when the accuracy of calculations is important.

When the reliability of the result is critical, an alternative to using floating-point numbers to approximate real numbers is to use real intervals instead. With intervals, one considers a range of possibilities instead of an exact value. Real interval arithmetic makes tracking rounding errors natural and robust. In addition to that, real intervals can also ease the representation of lack of knowledge in the exact value measures. For example, it is often the case that a value cannot be measured exactly, but it is possible to measure the value to be inside of some interval.

2.1.1 Definitions

Definition 2.1.1 (Interval)**.** A *real interval* or *interval* $\mathbf{i}$ is defined as a set of real numbers lying between (and including) given upper and lower bound; that is,

$$\mathbf{i} = [a,b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}.$$

The set of all real intervals is denoted by $\mathbb{IR}$.

For an interval $\mathbf{i} = [a,b] \in \mathbb{IR}$, a is called the *lower bound* while b is called the *upper bound*. Both lower and upper bound are also called *endpoints* of the interval.

We define the *width*, *midpoint* and *magnitude* of an interval $[a,b] \in \mathbb{IR}$ as

$$\begin{array}{lrcl} \text{Width:} & W([a,b]) & = & b-a, \\ \text{Midpoint:} & Mid([a,b]) & = & \frac{a+b}{2}, \\ \text{Magnitude:} & Mag([a,b]) & = & max\{|a|,|b|\}. \end{array}$$

Interval $\mathbf{i}$ is called *degenerate* if $W(\mathbf{i}) = 0$ and *symmetric* if $Mid(\mathbf{i}) = 0$.

A real interval vector

$$\vec{\mathbf{i}} = \begin{bmatrix} \mathbf{i}_1 \\ \mathbf{i}_2 \\ \vdots \\ \mathbf{i}_m \end{bmatrix} \in \mathbb{IR}^n$$

has n real interval components and can be interpreted as and n-dimensional *rectangle* or a *box*.

Given two interval vectors $\vec{\mathbf{i}}, \vec{\mathbf{j}} \in \mathbb{IR}^n$, we use $\vec{\mathbf{i}} \subseteq \vec{\mathbf{j}}$ to denote that $\vec{\mathbf{i}}[i] \subseteq \vec{\mathbf{j}}[i]$ for all $1 \leq i \leq n$. For an interval vector $\vec{\mathbf{i}} \in \mathbb{IR}^n$, the width and midpoint are defined component-wise, i.e.

$$\begin{array}{lrcl} \text{Width:} & W(\vec{\mathbf{i}})[i] & = & W(\vec{\mathbf{i}}[i]), \\ \text{Midpoint:} & Mid(\vec{\mathbf{i}})[i] & = & Mid(\vec{\mathbf{i}}[i]) \end{array}$$

for all $1 \leq i \leq n$.

The magnitude of a interval vector is defined as the maximum magnitude of the components

$$\text{Magnitude:} \quad Mag(\vec{\mathbf{i}}) = \max_{1 \leq j \leq n} \{Mag(\vec{\mathbf{i}}[i])\}.$$

In the rest of the thesis, we also call real interval vectors simply interval vectors or vectors. For simplicity, we sometimes also use a Cartesian product to denote an interval vector.

A real interval matrix can be defined as we defined an interval vector. Similarly, for a real interval matrix, the width and midpoint are defined component-wise, the magnitude is the maximum magnitude of the elements of the matrix. We will call real interval matrices simply interval matrices or just matrices in the rest of the thesis.

2.1.2 Operations and Functions

When dealing with sets as inputs to functions, we cannot just compute the function for each of the inputs. The problem with that is that sets can (and usually do) contain infinitely many elements. We need to instead introduce set based function evaluation.

Definition 2.1.2 (United extension)**.** Given a function $f : D \mapsto \mathbb{R}$ for $D \subseteq \mathbb{R}^n$ and some $n \in \mathbb{N}$. We call a set-valued function $F : 2^D \mapsto 2^{\mathbb{R}}$ the *united extension* of f if

$$F(\vec{C}) = \{f(\vec{c}) | \vec{c} \in \vec{C}\}$$

for all $\vec{C} \in 2^D$.

Unfortunately, if the sets are restricted to intervals, then the united extension of the function does not exist in general and we have to resort to a weaker version of the united extension called *interval extension*, which produces the same result as f on degenerate intervals.

Definition 2.1.3 (Interval extension)**.** Given a function $f : D \mapsto \mathbb{R}$ for a set $D \subseteq \mathbb{R}^n$ and some $n \in \mathbb{N}$. We call a interval-valued function $F : \mathbb{IR}^n \mapsto \mathbb{IR}$ the *interval extension* of f if for all $\vec{c} \in D$ there is

$$F(\vec{C}) = \{f(\vec{c}) | \vec{c} \in \vec{C}\}$$

where $\vec{C} = [\vec{c}, \vec{c}]$.

Definition 2.1.4 (Inclusion isotonicity)**.** Interval extension F over the domain $D \subseteq \mathbb{IR}^n$ is called inclusion isotonic, if for all $\vec{X}, \vec{Y} \in D$ and $\vec{X} \subseteq \vec{Y}$, it follows that $F(\vec{X}) \subseteq F(\vec{Y})$.

The fundamental theorem of interval analysis gives us a way to use intervals as inputs to a function. It guarantees that if an interval extension F of a real-valued function f is inclusion isotonic, then for any interval input $\vec{C} \subseteq Dom(f)$, $F(\vec{C})$ is an interval enclosure and $F(\vec{C})$ is an over-approximation of the exact function range $\{f(\vec{c}) | \vec{c} \in \vec{C}\}$.

Theorem 2.1.1 (Fundamental theorem of interval analysis [MKC09])**.** If F is an inclusion isotonic interval extension of f, then we have that $\{f(\vec{c}) | \vec{c} \in \vec{C}\} \subseteq F(\vec{C})$ for all interval inputs $\vec{C} \subseteq Dom(f)$.

If f is a continuous function, then the interval extension of f is also continuous. Hence, the basic four arithmetic operators $+, -, \cdot, /$ over reals can be extended to deal with intervals, the endpoints of the results can be computed from the endpoints of the operands.

$$\begin{array}{lrcl}
\text{Addition:} & [a,b]+[c,d] & = & [a+c,b+d] \\
\text{Subtraction:} & [a,b]-[c,d] & = & [a-d,b-c] \\
\text{Multiplication:} & [a,b][c,d] & = & [\min\{ac,ad,bc,bd\},\max\{ac,ad,bc,bd\}] \\
\text{Division:} & [a,b]/[c,d] & = & [a,b][\frac{1}{d},\frac{1}{c}] \text{ if } 0 \notin [c,d]
\end{array}$$

The addition and multiplication operators on intervals are *commutative* and *associative*. In addition, multiplication is *sub-distributive* over addition on intervals i.e. for intervals $\mathbf{i}_1$, $\mathbf{i}_2$ and $\mathbf{i}_3$, we have the following inclusion:

$$\mathbf{i}_1(\mathbf{i}_2+\mathbf{i}_3) \subseteq \mathbf{i}_1\mathbf{i}_2+\mathbf{i}_1\mathbf{i}_3$$

in which the equivalence does not generally hold.

It is possible to extend many standard functions to their interval counterparts. If the function is monotonic then the interval extension is simply applying the function to the endpoints:

$$\begin{array}{lrcl}
\text{Monotonically increasing:} & f([a,b]) & = & [f(a),f(b)] \\
\text{Monotonically decreasing:} & f([a,b]) & = & [f(b),f(a)].
\end{array}$$

If the function is not monotonic, then we need a custom algorithm. As an example, let us consider the interval extension of the sine function. With the sine function, we need to consider whether the minimum and the maximum are achieved on the interval used as the argument to the interval extension. The full algorithm for computing the interval extension for the sine function is given in Algorithm 1 ([Che15]). Note that the upper and lower endpoints of the interval extension are not necessarily computed from the endpoints of the argument.

Where it is possible, we can get an interval extension of a function by applying interval extension of the operations present in the function. That way the rigorous bound information is carried through the operations and in the end we obtain rigorous bounds of the function itself. While this is fast in practice, it is usually less precise than specifically tailored algorithms. As an example of this one can look for computing the n^{th} power of an interval (Algorithm 1) in [Che15].

2.1.3 Interval Arithmetic in Practice

The presentation of the interval arithmetic that we provided can be improved in practice in 2 aspects:

Algorithm 1: Interval extension of the sine function.

Input : An interval $[a,b]$

Output: the interval of $sin([a,b])$

if $\exists n \in \mathbb{Z}.((2n-\frac{1}{2})\pi \in [a,b])$ **then**
    c = -1 ;
else
    c = $\min\{sin(a), sin(b)\}$;
end
if $\exists n \in \mathbb{Z}.((2n+\frac{1}{2})\pi \in [a,b])$ **then**
    d = 1 ;
else
    d = $\max\{sin(a), sin(b)\}$;
end
return [c,d] ;

1. subverting restrictions
2. efficiency

Regarding 1, our exposition of interval arithmetic has three kinds of restrictions:

- we disallow division by zero,
- we disallow unbounded intervals,
- we are only considering intervals that are closed.

We refer to [HJVE01] on how to overcome all of these restrictions but mention that none of the restrictions affects the class of problems we are interested in.

Regarding 2, it is possible to exploit certain details of floating-point arithmetic to speed-up computations with interval arithmetic.

As an example of this, we will discuss multiplication.

To do so, we will first divide the intervals into classes based on the signs of the endpoints (Table 2.1).

We will also have 2 rounding modes for multiplication (respectively division), the rounding down is denoted by $*_{lo}$ ($/_{lo}$) and rounding up is denoted by $*_{hi}$ ($/_{hi}$). For a subset of real numbers $\alpha \subset \mathbb{R}$, we will use $\Gamma(\alpha)$ to denote the smallest floating-point interval containing it. The optimal multiplication of IEEE floating-point intervals is

Class of [a,b]	at least one negative	at least on positive	signs of endpoints
M	yes	yes	$a<0 \wedge b>0$
Z	no	no	$a=0 \wedge b=0$
P	no	yes	$a\geq 0 \wedge b>0$
P_0	no	yes	$a=0 \wedge b>0$
P_1	no	yes	$a>0 \wedge b>0$
N	yes	no	$a<0 \wedge b\leq 0$
N_0	yes	no	$a<0 \wedge b=0$
N_1	yes	no	$a<0 \wedge b<0$

Table 2.1: Classification of intervals by sign.

given in Table 2.2. As can be seen, besides one case, we only need to do one multiplication (instead of 4) to determine the endpoint of the smallest interval containing the result. In the one exceptional case, we have to do 2 multiplications, which is still better than 4.

The division (with or without allowing division by zero), can be handled similarly. The division also benefits from the fact, that for any floating-point numbers a and b, the inequalities $a *_{lo} (1/_{lo} b) \leq a/_{lo} b$ and $a *_{hi} (1/_{hi} b) \geq a/_{hi} b$ hold.

Class of $[a,b]$	Class of $[c,d]$	$\Gamma([a,b][c,d])$
P	P	$[a *_{lo} c, b *_{hi} d]$
M	P	$[a *_{lo} d, b *_{hi} d]$
N	P	$[a *_{lo} d, b *_{hi} c]$
P	M	$[b *_{lo} c, b *_{hi} d]$
M	M	$[\min(a *_{lo} d, b *_{lo} c), \max(b *_{hi} d, a *_{hi} c)]$
N	M	$[a *_{lo} d, a *_{hi} c]$
P	N	$[b *_{lo} c, a *_{hi} d]$
M	N	$[b *_{lo} c, a *_{hi} c]$
N	N	$[b *_{lo} d, a *_{hi} c]$
Z	any	$[0,0]$
any	Z	$[0,0]$

Table 2.2: Multiplication of IEEE intervals.

2.2 Taylor Models

Similarly, as intervals can be used to over-approximate real numbers, Taylor models can be used to over-approximate functions [Apo67, Apo69].

2.2.1 Taylor Approximation

A k-times differentiable univariate function f over a domain $D \subseteq \mathbb{R}$ can be approximated by a Taylor polynomial.

Definition 2.2.1 (Univariate Taylor approximation)**.** Given a univariate function $f : D \mapsto \mathbb{R}$, that is k-times differentiable at the point $x_0 \in D \subseteq \mathbb{R}$. We call

$$p_k(x) = \sum_{i=0}^{k} \frac{1}{i!} f^{(i)}(x_0)(x - x_0)^i \tag{2.1}$$

the order k Taylor approximation of f at the point x_0.

Example 1 (The sine function approximation)**.** The sine function can be approximated with an order k Taylor approximation at point 0 with

$$sin(x) \sim \sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

If f is $(k+1)$-times differentiable at the point x_0, then the error of the approximation $p_k(x)$ for any $x \in D$ can be quantified by univariate *Lagrange remainder term*

$$r_k(x) = f(x) - p_k(x) = \frac{1}{(k+1)!} f^{(k+1)}(x_0 + (x - x_0)\zeta(x))(x - x_0)^{(k+1)} \tag{2.2}$$

for some constant $0 < \zeta(x) < 1$.

If $f \in \mathcal{C}^{\omega}(D)$ is infinitely times differentiable at the point x_0, then there is some $\varepsilon > 0$ such that for any $x \in D$ if $|x - x_0| < \varepsilon$ that $p_k(x)$ converges to $f(x)$ when $k \to \infty$.

Definition 2.2.2 (Multivariate Taylor approximation)**.** Given a multivariate function $f : D \mapsto \mathbb{R}$, that is k-times differentiable at a point $\vec{x_0} \in D \subseteq \mathbb{R}^n$. We call

$$p_k(\vec{x}) = \sum_{i=0}^{k} \frac{1}{i!} [(\vec{x} - \vec{x}_0) \cdot \nabla]^i f(\vec{x}_0) \tag{2.3}$$

where the partial differential operator $[\vec{x}\cdot\nabla]^i$ is

$$[\vec{x}\cdot\nabla]^i = \sum_{\substack{j_1+\cdots+j_m=i \\ 0\leq j_1,\ldots,j_m\leq i}} \frac{k!}{j_1!\cdots j_m!}\vec{x}[1]^{j_1}\cdots\vec{x}[m]^{j_m}\frac{\partial^i}{\partial x_1^{j_1}\cdots x_m^{j_m}}$$

the order k Taylor approximation of f at the point $\vec{x}_0$.

Like in the univariate case, if the f is $(k+1)$-times partially differentiable at the point $\vec{x}_0$, we can bound the error of the approximation for any $\vec{x}\in D$ by Lagrange remainder term

$$r_k(\vec{x}) = f(\vec{x}) - p_k(\vec{x}) = \frac{1}{(k+1)!}[(\vec{x}-\vec{x}_0)\cdot\nabla]^{k+1} f(\vec{x}_0+(\vec{x}-\vec{x}_0)\zeta(\vec{x})) \tag{2.4}$$

for some constant $0<\zeta(\vec{x})<1$.

If $f\in\mathcal{C}^\omega(D)$ is infinitely times differentiable at the point $\vec{x}_0$, then there is some $\varepsilon>0$ such that for any $\vec{x}\in D$ if $||\vec{x}-\vec{x}_0||<\varepsilon$ that $p_k(\vec{x})$ converges to $f(\vec{x})$ when $k\to\infty$.

It is possible to find the Taylor polynomial of a composition of functions given the Taylor polynomials of the functions used in the composition. Given two functions $f:\mathbb{R}^n\mapsto X$ and $g:Y\mapsto\mathbb{R}^m$, where $X\subseteq Y$. Let us assume that we have order k Taylor polynomials for each of these - $p_f(\vec{x})$ for f at the point $\vec{c}$ and $p_g(\vec{y})$ for g at the point $f(\vec{c})$. Then the order k Taylor polynomial for the composite function $g\circ f:\mathbb{R}^n\mapsto\mathbb{R}^m$ can be found by doing the substitution $\vec{y}\mapsto p_f(\vec{x})$ in p_g and discarding the terms[1] whose degree is higher than k.

Example 2 (Composition of Taylor polynomials)**.** Suppose we have an order 3 Taylor approximation for sine function $sin(x)\sim x-\frac{x^3}{3!}$ and order 3 Taylor approximation for cosine function $cos(x)\sim 1-\frac{x^2}{2!}$, both at point 0. The order 3 Taylor approximation of the composition $cos\circ sin$ can be computed by substituting the approximation for sin into the approximation for cos and discarding the terms with the degree higher than 3

$$(cos\circ sin)(x)\sim 1-\frac{1}{2!}\left(x-\frac{x^3}{3!}\right)^2 = 1-\frac{x^2}{2}+\frac{x^4}{3}-\frac{x^6}{18}\sim 1-\frac{x^2}{2}$$

2.2.2 Functions as Taylor Models

Definition 2.2.2 gives us a way to represent functions in a way that we get an over-approximation. The idea was originally developed by Berz and Makino [Mak98, MB11]. The gist of it is to approximate functions with polynomials and enclose approximation errors using intervals.

[1] Note that we are talking about composition of Taylor polynomials here, not composition of Taylor models which we define and address later.

Definition 2.2.3 (Taylor model)**.** Given a n-variable function $f : D \mapsto \mathbb{R}$, where $D \subseteq \mathbb{R}^n$. We call the pair $(p, \mathbf{i})$, where p is n-variable polynomial and $\mathbf{i} \in$ is an interval, a *Taylor model* for the function f if

$$f(\vec{x}) \in p(\vec{x}) + \mathbf{i}$$

holds for all $\vec{x} \in D$.

We make it explicit that the domain of $(p, \mathbf{i})$ is D by using the notation $(p, \mathbf{i}) \bullet D$.

Sometimes we refer to Taylor models by using tuples of polynomials and remainders, other times we sum up the two parts, i.e. if the polynomial is p and the interval remainder is $\mathbf{i}$ we may use $(p, \mathbf{i})$ and $p + \mathbf{i}$, respectively.

We will use letters $\mathcal{T}, \mathcal{U}, \mathcal{V}$ to denotes Taylor models. We use *poly* and *rem* to refer to the polynomial and the remainder interval of the Taylor model. That is, if $\mathcal{T} = (p, \mathbf{i})$, then $\mathcal{T}.poly = p$ and $\mathcal{T}.rem = \mathbf{i}$.

The Taylor model for a function $f : D \mapsto \mathbb{R}$, where $D \subseteq \mathbb{R}^n$ can be viewed as a function $\mathcal{T} : D \mapsto \mathbb{IR}$, but also as a function $\mathcal{T} : \mathbb{IR}^n \mapsto \mathbb{IR}$. In addition to viewing Taylor models as functions, we may also view them as sets i.e. the images of their domains which we call *ranges* of Taylor models.

Definition 2.2.4. Given a Taylor model $\vec{\mathcal{T}} = p(\vec{a}) + \mathbf{i}$ with a domain D we define its *range* as the set

$$Range(\vec{\mathcal{T}}, D) = \bigcup_{\vec{a} \in D} p(\vec{a}) + \mathbf{i}$$

A vector-valued function $\vec{f} : D \mapsto \mathbb{R}^m$, where $D \subseteq \mathbb{R}^n$, can be viewed as a vector of real-valued functions $\vec{f}[i]$ for $i \in \{1, \ldots, n\}$. For any of these real-valued functions $\vec{f}[i]$, we can find a Taylor model $(p_i, \mathbf{i}_i)$. Composing these Taylor models into a vector, we can get a Taylor model, denoted as $\vec{\mathcal{T}} : \mathbb{IR}^n \mapsto \mathbb{IR}^m$, for the vector-valued function $\vec{f}$ over D.

When we will use vectors of Taylor models we will often name the corresponding elements of these vectors, supposing the name of the element is x, we refer to the element of Taylor model $\vec{\mathcal{T}}$ corresponding to that element as $\mathcal{T}_x$.

For reasons becoming clear later, we associate elements of vector Taylor models with variables, to avoid confusion with the variables of the polynomials appearing in the Taylor models we refer to to the variables of the polynomials as *parameters* and elements of vectors of Taylor models as *elements corresponding to variables*, *elements*

associated with variables or just *variables*. From this point onwards we will use letters x, y and z to refer to variables and letters a and b to refer to parameters.

Example 3 (Taylor model terminology)**.** Suppose we have a vector of Taylor models $\vec{\mathcal{T}}$ with domain D. Suppose that the first element corresponds to variable x_1 and the second element corresponds to variable x_2. Suppose that the element corresponding to x_1 has p_{x_1} as the polynomial and $\mathbf{i}_{x_1}$ as the interval. Likewise, suppose that the element corresponding to x_2 has p_{x_2} as the polynomial and $\mathbf{i}_{x_2}$ as the interval. We may represent this vector of Taylor models as

$$\vec{\mathcal{T}} = \begin{bmatrix} \mathcal{T}_{x_1} \\ \mathcal{T}_{x_2} \end{bmatrix} = \begin{bmatrix} (p_{x_1}, \mathbf{i}_{x_1}) \\ (p_{x_2}, \mathbf{i}_{x_2}) \end{bmatrix} = \begin{bmatrix} p_{x_1} + \mathbf{i}_{x_1} \\ p_{x_2} + \mathbf{i}_{x_2} \end{bmatrix}$$

Let us assume that the variables of polynomials p_{x_1} and p_{x_2} are named a_1 and a_2. Then we say that the Taylor model $\vec{\mathcal{T}}$ has elements x_1 and x_2 and parameters a_1 and a_2.

Sometimes we represent a single Taylor model using a composition of two Taylor models.

Definition 2.2.5 (Factored Taylor model)**.** The composition

$$\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$$

of the two Taylor models $\vec{\mathcal{U}}$ and $\vec{\mathcal{V}}$ is called a *factored Taylor model* $\vec{\mathcal{T}}$ if

(i) $D = D_r$ where D is the domain of $\vec{\mathcal{T}}$ and D_r is the domain of $\vec{\mathcal{V}}$,

(ii) $Range(\vec{\mathcal{V}}, D_r) \subseteq D_l$ where D_l is the domain of $\vec{\mathcal{U}}$,

(iii) $\vec{\mathcal{T}}(\vec{a}) \subseteq (\vec{\mathcal{U}} \circ \vec{\mathcal{V}})(\vec{a})$ for all $\vec{a} \in D$.

Taylor model $\vec{\mathcal{U}}$ is called the *left Taylor model* or just the *left model* and Taylor model $\vec{\mathcal{V}}$ is called the *right Taylor model* or just the *right model*.

The composition $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ of Taylor models $\vec{\mathcal{U}}$ and $\vec{\mathcal{V}}$ is to be understood as the insertion of the Taylor models corresponding to the elements of $\vec{\mathcal{V}}$ into the parameters of the left Taylor model $\vec{\mathcal{U}}$.

The factoring of the Taylor model is not unique. Two different factored Taylor models can represent the same Taylor model, but use different polynomials and intervals to do that.

Example 4 (Factored Taylor model)**.** Suppose we have a Taylor model $\vec{\mathcal{T}} = c + a + a^2 + \mathbf{i}$ where the domain is D, c is scalar and a is the parameter of the Taylor model.

We may represent this Taylor model with factored Taylor model $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ where $\vec{\mathcal{U}} = c + a + [0,0]$, $\vec{\mathcal{V}} = a + a^2 + \mathbf{i}$ where the domain of $\vec{\mathcal{V}}$ is D.

Or we may represent this Taylor model with another factored Taylor model $\vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'$ where $\vec{\mathcal{U}}' = c + a + a^2 + [0,0]$, $\vec{\mathcal{V}}' = a + \mathbf{i}$ and the domain of $\vec{\mathcal{V}}'$ is D. Note that if $\mathbf{i} \neq [0,0]$, then $Range(\vec{\mathcal{U}} \circ \vec{\mathcal{V}}, D) \subset Range(\vec{\mathcal{U}}' \circ \vec{\mathcal{V}}', D)$.

Sometimes we make the parameters of Taylor model explicit by writing them in parentheses i.e. if Taylor model $\vec{\mathcal{T}}$ has parameters a_1 and a_2 we may make it explicit by using the name $\vec{\mathcal{T}}(a_1, a_2)$ instead of $\vec{\mathcal{T}}$. We treat polynomials similarly.

Definition 2.2.6 (Convex Set)**.** A set S is convex if and only if for all $x, y \in S$, it is satisfied that

$$\lambda x + (1 - \lambda) y \in S$$

for all $\lambda \in [0,1]$.

Definition 2.2.7 (Normed vector space)**.** A normed vector space $(M, |\cdot|)$ is a vector space M equipped with a norm $|\cdot|$.

Definition 2.2.8 (Bounded Set)**.** A set S in a normed vector space $(M, |\cdot|)$ is bounded, if there exists $r > 0$ such that $|x - y| < r$ for all $x, y \in S$.

Definition 2.2.9 (Open set)**.** A set S in a normed vector space $(M, |\cdot|)$ is open, if for any $x \in S$ there exists $\varepsilon > 0$ such that for any $y \in S$ if $|x - y| < \varepsilon$ then $y \in S$.

Definition 2.2.10 (Closed Set)**.** A set S is closed if its complement is open.

Definition 2.2.11 (Compact Set)**.** A set S is compact if it is bounded and closed.

Theorem 2.2.1. A TM over an interval domain D defines a convex and compact set of continuous functions which are over-approximated by it over D [BM98].

2.2.3 Taylor Model Arithmetic

We need to define the same operations on Taylor models as what we are applying to the functions if we want to use Taylor models in place of functions.

Typically, Taylor model arithmetic implementations place limits on the size of the Taylor models. This is usually done by placing limits on the total order of terms appearing in the Taylor model or by limiting the order of each parameter individually.

When computations produce Taylor models with terms beyond these limits, the Taylor model polynomials are truncated and interval bounds on the removed terms are added to the Taylor model interval. Adjustment of this limit is used to tradeoff accuracy and computation time.

Definition 2.2.12 (Interval bound of a Taylor polynomial)**.** Given a n-dimensional vector-valued function $\vec{f}$ and a domain D, we call $B(\vec{f}) \subseteq \mathbb{IR}^n$ the interval bound of $\vec{f}$ if

$$\vec{f}(\vec{x}) \in B(\vec{f})$$

holds for all $\vec{x} \in D$.

Definition 2.2.13 (Addition of Taylor models)**.** Given vector-valued functions $\vec{f} : D \mapsto \mathbb{R}^m$ and $\vec{g} : D \mapsto \mathbb{R}^m$, where $D \subseteq \mathbb{R}^n$ with their respective order k Taylor models $\vec{\mathcal{T}}_f = (\vec{p}_f, \vec{\mathbf{i}}_f)$ and $\vec{\mathcal{T}}_g = (\vec{p}_g, \vec{\mathbf{i}}_g)$.

We define the order k Taylor model of the function $f + g$ as

$$\vec{\mathcal{T}}_{f+g} = (\vec{p}_f + \vec{p}_g, \vec{\mathbf{i}}_f + \vec{\mathbf{i}}_g)$$

over the domain D

The addition of Taylor models is both commutative and associative.

Definition 2.2.14 (Additive inverse)**.** Given a Taylor model $\vec{\mathcal{T}}_f = (\vec{p}_f, \vec{\mathbf{i}}_f)$ for a function $\vec{f}$ over the domain D, we call $(-\vec{p}_f, -\vec{\mathbf{i}}_f)$ over the same domain D, its additive inverse. This is also a Taylor model for the function $-\vec{f}$ over the domain D.

The multiplication of Taylor models is slightly more intricate. The problem is that in multiplying polynomials we get polynomials of higher-order than the required k. To overcome it we need to bound these terms with intervals and over-approximate the multiplication.

Definition 2.2.15 (Multiplication of Taylor models)**.** Given vector-valued functions $\vec{f} : D \mapsto \mathbb{R}^m$ and $\vec{g} : D \mapsto \mathbb{R}^m$, where $D \subseteq \mathbb{R}^n$ with their respective order k Taylor models $\vec{\mathcal{T}}_f = (\vec{p}_f, \vec{\mathbf{i}}_f)$ and $\vec{\mathcal{T}}_g = (\vec{p}_g, \vec{\mathbf{i}}_g)$.

We define the order k Taylor model of the function $f \times g$ over the domain D as

$$\vec{\mathcal{T}}_{f\times g} = (\vec{p}_{f\times g}, \vec{\mathbf{i}}_{f\times g})$$

where

- the product $\vec{p}_f \times \vec{p}_g$ is split into two parts: $\vec{p}_{f\times g}$ (containing the all the terms of order k or less) and p_e (terms with order higher than k)
- $\vec{\mathbf{i}}_{f\times g} = B(\vec{p}_e) + B(\vec{p}_f) \times \vec{\mathbf{i}}_g + B(\vec{p}_g) \times \vec{\mathbf{i}}_f + \vec{\mathbf{i}}_f \times \vec{\mathbf{i}}_g$.

Definition 2.2.16 (Remainders of Taylor models)**.** Given an n dimensional vector of Taylor models

$$\vec{\mathcal{T}} = \begin{bmatrix} p_1 & + & \mathbf{i}_1 \\ & \vdots & \\ p_n & + & \mathbf{i}_1 \end{bmatrix}$$

we will use $rem(\vec{\mathcal{T}})$ to denote the vector of its remainders

$$rem(\vec{\mathcal{T}}) = (\mathbf{i}_1, \ldots, \mathbf{i}_n)^T$$

A common way to find a Taylor model for a function is to use truncated Taylor series for that function as the polynomial and the error bound of the Taylor series on the domain as the remainder interval.

Example 5 (Taylor model for sine function)**.** Suppose we want to find an order 5 Taylor model for sine function with domain $[0,1]$.

We can use Example 1 to find a polynomial part of the Taylor model for sine function as

$$a - \frac{a^3}{6} + \frac{a^5}{120}$$

From (2.2) we know that the error can be bounded with

$$-sin(a\zeta(a))\frac{a^6}{720}$$

for some constant $0 < \zeta(a) < 1$.

We can bound the error with an interval by substituting parameter a with its domain and the constant $\zeta(a)$ with an interval $[0,1]$ which contains all possible values for it, i.e. we get

$$\begin{aligned} -sin([0,1][0,1])\tfrac{[0,1]^6}{720} &= -sin([0,1])\tfrac{[0,1]}{720} = -[0,sin(1)][1,\tfrac{1}{720}] = -[0,sin(1)] \\ &= [-sin(1),0] \end{aligned}$$

That is, sine function can be represented with an order 5 Taylor model

$$\vec{\mathcal{T}} = a - \frac{a^3}{6} + \frac{a^5}{120} + [-sin(1),0]$$

on domain $D = [0,1]$.

2.2.4 Parameter Dependencies in the System

We use the system variables as the names of the system's dimensions. We do this in order to better relate the elements of the Taylor models with their derivatives, solutions and flows[2]. The following functions have their domain over these names. That is, they are not real-valued functions of the system variables.

Definition 2.2.17 (Initial condition parameters of a variable)**.** Given an n-dimensional system where the initial conditions for variables are expressed as Taylor models over parameters $\vec{a}$

$$\begin{aligned} x_1(0) &= g_1(\vec{a}) + \mathbf{i}_1 \\ &\vdots \\ x_n(0) &= g_n(\vec{a}) + \mathbf{i}_n \end{aligned}$$

We define the initial condition parameters of the variable x_i as

$$ip(x_i) = \{a \in \vec{a} \mid g_i \text{ depends on } a\}$$

Definition 2.2.18 (Left model parameter)**.** For a system with n variables we associate a unique *left model parameter* with each of the variables using a function pv. We require that each variable has a unique parameter i.e. if $pv(x_i) = b_i$ and $pv(x_j) = b_j$ then $x_i = x_j \Leftrightarrow b_i = b_j$. We call the set of all left model parameters $\vec{b}$. We also require pv to be a total function and we denote its inverse as pv^{-1}.

Definition 2.2.19 (Initial conditions of variables)**.** Given n variables $x_1, \ldots, x_n$ and n initial conditions $x_1(0), \ldots, x_n(0)$, we associate the i^{th} variable with i^{th} initial condition with the function *init*

$$init(x_i) = x_i(0)$$

When it is clear from the context we will use the same functions to associate a set of variables with a set of initial conditions

$$init(X) = \{init(x) \mid x \in X\}$$

2.3 Chemical Reaction Networks

A chemical reaction network (CRN) is a tuple $\mathcal{C} = (\mathcal{S}, \mathcal{R})$, where $\mathcal{S} = \{s_0, \ldots, s_{n-1}\}$ and $\mathcal{R} = \{r_0, \ldots, r_{m-1}\}$ denote the finite sets of species and reactions, respectively. A

[2] We define *solution* and *flow* in Chapter 3.

reaction is a tuple $r = (\mathbf{r}^r, \mathbf{p}^r, k^r)$ where $\mathbf{r}^r \subset \mathbb{N} \times \mathcal{S}$ and $\mathbf{p}^r \subset \mathbb{N} \times \mathcal{S}$ are the reactants and products, k denotes the reaction propensity prefactor of r. The concentration of species s at time t is denoted by $c(s,t) \in \mathbb{R}$.

The propensity of the reaction r is defined as $pro_r(t) = k \prod_{s \in \mathbf{r}^r} c(s,t)$. The semantics of propensity is that in at time t $pro_r(t)$ amount of reactants are consumed while the same amount of products are produced.

For reaction $r = (\mathbf{r}^r, \mathbf{p}^r, k^r)$ we will often use the notation

$$r : \sum_{(j,s) \in \mathbf{r}^r} js \xrightarrow{k_r} \sum_{(j,s) \in \mathbf{p}^r} js.$$

Given 2 reactions $r_1 = (\mathbf{r}^{r_1}, \mathbf{p}^{r_1}, k^{r_1})$ and $r_2 = (\mathbf{r}^{r_2}, \mathbf{p}^{r_2}, k^{r_2})$ where $\mathbf{r}^{r_1} = \mathbf{p}^{r_2}$ and $\mathbf{r}^{r_2} = \mathbf{p}^{r_1}$ we will combine the two reactions as

$$r : \sum_{(j,s) \in \mathbf{r}^{r_1}} js \underset{k^{r_2}}{\overset{k^{r_1}}{\rightleftharpoons}} \sum_{(j,s) \in \mathbf{p}^{r_1}} js.$$

Example 6 (CRN)**.** Suppose we have a CRN with 4 species X_1, X_2, X_3, X_4 and 4 reactions r_1, r_2, r_3, r_4. The reactants of r_1, r_2, r_3 and r_4 are the sets $\{X_1, X_2\}$, $\{X_3\}$, $\{X_1\}$, $\{X_3\}$, respectively. The products of r_1, r_2, r_3 and r_4 are the sets $\{X_3\}$, $\{X_1, X_2\}$, $\{X_4\}$, $\{X_4\}$, respectively. The propensities of reactions r_1, r_2 and r_3 are $k_{r_1}, k_{r_2}, k_{r_3}, k_{r_4}$, respectively

We use

$$\begin{aligned} r_{1,2} &: X_1 + X_2 \underset{k_{r_1}^{\leftarrow}}{\overset{k_{r_2}^{\rightarrow}}{\rightleftharpoons}} X_3 \\ r_3 &: X_1 \xrightarrow{k_{r_3}^{\rightarrow}} X_4 \\ r_4 &: X_3 \xrightarrow{k_{r_4}^{\rightarrow}} X_4 \end{aligned}$$

to describe this CRN.

Chapter 3

Validated Integration

3.1 Introduction

Validated integration computes enclosures guaranteed to contain the solutions of systems of ODEs. In the simplest form, an enclosure is computed for a point initial condition. More generally, the enclosure is computed for (closed bounded) sets of initial conditions.

The enclosure is computed in three phases: (i) approximation of the solution is computed, (ii) the approximation is enlarged until it contains the solution, (iii) the enlargement is minimized while still guaranteeing that it is a valid enclosure of the solution.

In the Taylor model based validated integration, techniques called shrink wrapping and preconditioning can be used between the integration of different time steps to condition the set to be integrated to introduce less over-approximation.

3.2 Problem Description

In this section, we present the problem we are trying to solve and introduce our notation. We begin by defining the initial value problem [Mei07].

Definition 3.2.1 (Initial Value Problem)**.** An initial value problem is the triple

$$\frac{d\vec{x}}{dt} = \vec{f}(\vec{x}), \quad \vec{x}_{init}, \quad [t_{init}, t_{final}]$$

where the vector field $\vec{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a sufficiently-smooth function, $\vec{x}_{init} \in \mathbb{R}^n$ is the initial condition and $[t_{init}, t_{final}]$ is the integration period.

A *solution*[1] to an initial value problem is a function $\vec{\phi}(t;\vec{x}_{init})$ that satisfies the conditions

(1) $\frac{d\vec{\phi}(t;\vec{x}_{init})}{dt} = \vec{f}(\vec{\phi}(t;\vec{x}_{init}))$ for $t \in [t_{init}, t_{final}]$

(2) $\vec{\phi}(t_{init};\vec{x}_{init}) = \vec{x}_{init}$

i.e. $\vec{\phi}(t;\vec{x}_{init})$ is a solution to the system of differential equations during the span of the integration and evaluates to initial condition at the start of the integration.

We give a wider definition of IVP because the validated integration approach works over sets of initial conditions. With this definition we may talk about a family of functions corresponding to the solution for each point in the set or we may also talk about the solution to the set itself.

Definition 3.2.2 (Extended Initial Value Problem)**.** An extended initial value problem is a triple

$$\frac{d\vec{x}}{dt} = \vec{f}(\vec{x}), \quad \vec{X}_{init}, \quad [t_{init}, t_{final}] \tag{3.1}$$

where the vector field $\vec{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a sufficiently-smooth function, $\vec{X}_{init} \subset \mathbb{R}^n$ is the set of initial condition and $[t_{init}, t_{final}]$ is the integration period.

A *solution* to an extended initial value problem is the family of functions

$$\vec{\phi}(\vec{X}_{init}) = \{\lambda t.\ \vec{\phi}(t;\vec{x}_{init}) \mid \vec{x}_{init} \in \vec{X}_{init}\}$$

where $\vec{\phi}(t;\vec{x}_{init})$ is the solution to IVP $(\vec{f}, \vec{x}_{init}, [t_{init}, t_{final}])$.

A fat solution is the function $\Phi : \mathbb{R} \mapsto 2^{R^n}$ defined as

$$\vec{\Phi}(\vec{X}_{init})(t) = \{\vec{\phi}(t;\vec{x}_{init}) \mid \vec{x}_{init} \in \vec{X}_{init}\}$$

In the following, we will always assume that for any $\vec{x}_{init} \in \vec{X}_{init}$ and any $t \in [t_{init}, t_{final}]$ the solution $\vec{\phi}$ for $\vec{x}$ exists and is unique. For example, if $\vec{f}$ is Lipschitz continuous in neighbourhoods of all $\vec{x}_{init} \in \vec{X}_{init}$, then the Picard-Lindelöf theorem [Mei07] ensures the existence of solutions in some closed time interval around t_{init}, and we assume that $[t_{init}, t_{final}]$ is a subset of this interval.

Definition 3.2.3 (Neighbourhood)**.** Given an point $\vec{c} \in \mathbb{R}^n$. A neighbourhood of $\vec{c}$ is a set $U \subseteq \mathbb{R}^n$ if there exists an open set $V \subseteq \mathbb{R}^n$ such that $\vec{c} \in V \subseteq U$.

[1] $\vec{\phi}$ is a function of time, denoting the solution to variables $\vec{x}$. Solution to IVP is parametrised by both $\vec{f}$ and $\vec{x}_{init}$, we only make $\vec{x}_{init}$ explicit in the notation of ϕ since $\vec{f}$ is always fixed.

Definition 3.2.4 (Lipschitz continuity)**.** We say that a function $\vec{f}(\vec{x})$ is Lipschitz continuous w.r.t. $\vec{x}$ in an open set C, if there exists a real constant $L \geq 0$ such that for any $\vec{c}_1, \vec{c}_2 \in C$ the following inequality holds

$$|\vec{f}(\vec{c}_1) - \vec{f}(\vec{c}_2)| \leq L|\vec{c}_1 - \vec{c}_2|.$$

Theorem 3.2.1 (Picard-Lindelöf Theorem)**.** Suppose $\frac{d\vec{x}}{dt}$ is Lipschitz continuous w.r.t. $\vec{x}$ in some open set $C \subseteq \mathbb{R}^n$. Then, for any $\vec{x}_{init} \in C$, there exists $\varepsilon > 0$ such that there exists a unique solution $\vec{\phi}(t; \vec{x}_{init})$ to the initial value problem on the interval $[t_{init} - \varepsilon, t_{init} + \varepsilon] \subseteq T$.

In the following, we will always assume that the vector field in EIVP is Lipschitz continuous w.r.t. the variables $\vec{x}$. This results in the fact that if a solution exists in the interval $[t_{init} - \varepsilon, t_{init} + \varepsilon]$ then it is also unique [Mei07].

Most IVPs (and EIVPs) that are interest to us do not have a closed-form solution. Therefore, for any given EIVP, our goal is not to find a solution, but an enclosure containing the solution which we will call the flow of the EIVP.

Definition 3.2.5 (An over-approximation of the flow)**.** Given an EIVP (3.1) we call the function $g : \mathbb{R} \to \mathbb{IR}^n$ a *an over-approximation of the flow* or *enclosure of the flow* of the EIVP if for any $\psi \in \vec{\phi}(\vec{X}_{init})$

$$\psi(t) \in g(t)$$

for all $t \in [t_{init}, t_{final}]$.

Observation 3.2.1. If $g(t)$ is flow for a given EIVP then

$$\vec{\Phi}(\vec{X}_{init})(t) \subseteq g(t)$$

for all $t \in [t_{init}, t_{final}]$.

From this point onwards we also use terms or *flow* or *flowpipe* to refer both the over-approximations of the flow as well as the exact flow.

In practice, the flow for EIVP is not constructed for the whole integration period at once. Instead, the integration period is partitioned into smaller intervals. A sequence of values $t_0 = t_{init} < t_1 < \ldots < t_m = t_{final}$, is used to define m intervals $[t_i, t_i + 1]$ (for $0 \leq i < m$) called *time steps*. Flow is then constructed for each of these time steps.

First, an EIVP is defined for the first time step. This new EIVP uses the same vector field and initial condition as the original EIVP, but instead of $[t_{init}, t_{final}]$ it uses $[t_0, t_1]$ for the integration period. After we have computed the flow for the first time

step we consider the value of the flow at the end of the first time step. We use this set as the initial condition to define another EIVP with the same vector field and $[t_1, t_2]$ as the integration period to compute the flow for the second time step. Continuing this pattern we can compute flows for all of the time steps. We can get the flow for the original EIVP by joining these flows together.

There are two reasons to partition the integration period into smaller intervals. First, it may be impossible to validate the existence and uniqueness of the solution without doing that and secondly, using smaller time step often results in less over-approximation introduced when computing flows.

In the following, we are going to ignore this partitioning[2] and we will assume that the EIVPs under consideration will only have a single time step. Our motivation for doing that is to simplify the presentation, since by doing that we do not need to burden the notation with time step annotations. We are justified in ignoring this detail since independent of whether we partitioning the time period or not, the task at hand is to compute flow for the EIVP.

In the following, we will use Δ to denote the integration period $[t_{init}, t_{final}]$.

3.3 Taylor model Flowpipe Construction

In the Taylor model based validated integration, we want to represent the flow of the EIVP with Taylor models. This also means that we will view the initial conditions as represented by Taylor models (both points and intervals can be seen as such).

Definition 3.3.1 (Taylor model flow)**.** Suppose that the initial condition $\vec{X}_{init}$ in EIVP (3.1) is given by a Taylor model $\vec{T}^0 \bullet D$. A Taylor model $\vec{T} \bullet (\Delta \times D)$ encloses the flow of this EIVP if

$$\vec{\Phi}(\vec{T}^0(\vec{a}))(t) \subseteq \vec{T}(t, \vec{a})$$

holds for any $t \in \Delta$ and any $\vec{a} \in D$.

From this point onwards we use flow to denote Taylor model flow where it is clear from the context.

Observation 3.3.1. If a Taylor model encloses an EIVP solution, then

$$\vec{\Phi}(Range(\vec{T}^0 \bullet D))(t) \subseteq \vec{T}(t, D)$$

[2] When required, we will discuss specifying the time parameter to create the initial condition for the next step.

holds for any $t \in \Delta$.

In here, $\vec{\mathcal{T}}(t,D)$ is defined as

$$\vec{\mathcal{T}}(t,D) = \bigcup_{\vec{a}\in D} \vec{\mathcal{T}}(t,\vec{a})$$

At the core of using the Taylor model based validated integration is the Picard operator. Under conditions to be explained shortly Picard operator takes some approximation to the IVP solution and creates a better approximation.

Definition 3.3.2 (Picard operator)**.** Given an IVP $(\vec{f},\vec{x}_{init},[t_{init},t_{final}])$ and an approximation of its solution $\vec{g}:\mathbb{R}\to\mathbb{R}^n$, the Picard operator is defined as

$$\mathbb{P}_{\vec{f}}(\vec{g})(t) = \vec{x}_{init} + \int_{t_{init}}^{t} \vec{f}(\vec{g}(\tau))d\tau$$

It can be shown that the fixed point of the Picard operator defined by $\vec{f}$ and $\vec{x}_{init}$ is unique and equal to the solution of the IVP.

We are also interested in looking at a version of $\mathbb{P}_{\vec{f}}$ where the functions g are defined using parameters of Taylor models. To make the dependency on parameters explicit we use extra arguments in g, that is we also use the Picard operator

$$\mathbb{P}_{\vec{f}}(\vec{g})(t,\vec{a}) = \vec{x}_{init}(\vec{a}) + \int_{t_{init}}^{t} \vec{f}(\vec{g}(\tau,\vec{a}))d\tau$$

Suppose that the initial condition for EIVP is represented with the Taylor model $\vec{\mathcal{T}}^0 \bullet D$ with parameters $\vec{a}$. We can compute a flow for EIVP by finding a Taylor model $\vec{\mathcal{T}} \bullet (\Delta \times D)$ with parameters $(t,\vec{a})$ such that there exists a function $\vec{g}$ that is a fixed point of $\mathbb{P}_{\vec{f}}$ and the condition $\vec{g}(t,\vec{a}) \in \vec{\mathcal{T}}(t,\vec{a})$ is satisfied for all $\vec{a} \in D$ and $t \in \Delta$.

Given a Taylor model we can verify whether it represents the flow or not by using an interval generalisation $\mathbb{P}^I_{\vec{f}}()$ of the basic operator $\mathbb{P}_{\vec{f}}$ that is defined w.r.t. an EIVP $(\vec{f},\vec{X}_{init},\Delta)$ and allows $\vec{x}_{init},\vec{f}$ and $\vec{g}$ to take on interval values in $\mathbb{IR}^n$ and allows $\vec{f}$ and $\vec{g}$ to take interval arguments in $\mathbb{IR}^n$ and $\mathbb{IR}\times\mathbb{IR}^n$ respectively ($\vec{g}$ takes an extra time argument).

To be more precise, after fixing $\vec{a} \in D$, continuous functions $\vec{\phi}(T^0(\vec{a}))$ with the norm $|\cdot|$ defined by

$$|\, g \,| = sup\{g(t) \mid g \in \vec{\phi}(T^0(\vec{a})), t \in \Delta\}$$

form a Banach space and $\lambda t.\ \vec{\mathcal{T}}(t,\vec{a})$ defines a a convex and compact set of continuous functions. Therefore, if $\mathbb{P}^I_{\vec{f}}(\vec{\mathcal{T}}) \subseteq \vec{\mathcal{T}}$, we can infer by the Schauder fixed point theorem that there exists a function $\vec{g} \in \vec{\mathcal{T}} \bullet (\Delta \times D)$ which is a fixed point of $\mathbb{P}_{\vec{f}}$.

Assuming that the polynomials of $\mathbb{P}^I_{\vec{f}}(\vec{\mathcal{T}})$ and $\vec{\mathcal{T}}$ are equal[3] we may detect the inclusion of the Taylor models by verifying the inclusion on the remainder terms.

Definition 3.3.3 (Convergence)**.** Suppose we have a normed vector space $(S, |\cdot|)$. An infinite sequence $s_1, s_2, s_3, \ldots$ (where $s_i \in S$) is said to *converge* to a point $s \in S$ if $|\, s_i - s \,| \to 0$ when $i \to \infty$.

Definition 3.3.4 (Cauchy sequence)**.** Suppose we a normed vector space $(S, |\cdot|)$. An infinite sequence $s_1, s_2, s_3, \ldots$ (where $s_i \in S$) is said to be *Cauchy sequence* if for any $\varepsilon > 0$, there exists $n \in \mathbb{N}^+$ such for all $i, j \geq n$ we have that $|\, s_i - s_j \,| < \varepsilon$.

Definition 3.3.5 (Schauder fixed point theorem)**.** Suppose we have a convex and compact set U in a Banach space $(S, |\cdot|)$ that is a normed vector space whose Cauchy sequences are all convergent to an element in S. Then a continuous mapping $f : U \to U$ has a fix point in U.

The Picard operator can also be used to compute successively better approximations to a solution or sets of solutions over the integration period. Applying Picard operator iteratively converges to the fixed point of Picard operator.

Definition 3.3.6 (Picard iteration)**.** Given and EIVP $(\vec{f}, \vec{X}_{init}, [t_{init}, t_{final}])$ the Picard iteration is defined as

$$\vec{g}^{(k+1)}(t) = \mathbb{P}_{\vec{f}}(\vec{g}^{(k)})(t) = \vec{X}_{init} + \int_{t_{init}}^{t} \vec{f}(\vec{g}^{(k)}(\tau))d\tau \quad \text{for } k \geq 0$$

where $\vec{g}^{(0)}(t) = \vec{X}_{init}$.

Observation 3.3.2. Given and EIVP $(\vec{f}, \vec{X}_{init}, [t_{init}, t_{final}])$, we may compute the k^{th}-order truncated Taylor series of the solution to the EIVP around the point t_{init} by computing the k^{th} Picard iteration.

Example 7. We illustrate Observation 3.3.2 with and example.

Suppose we have an EIVP with one variable x, whose derivative is

$$\frac{dx}{dt} = 2x$$

and initial condition is $x(t_{init}) = 1$. Suppose also that our goal is to integrate for the period the period $[t_{init}, t_{final}]$).

[3] In general, $\mathbb{P}^I_{\vec{f}}(\vec{\mathcal{T}})$ has more terms than $\vec{\mathcal{T}}$. We can get rid of these extra terms by pushing them into the remainder interval. We will see later than this is happening automatically when operating on k-order Taylor model arithmetic.

We can compute the Picard iteration for this EIVP. Let us compute 3 iterations of it.

$$\begin{aligned}
x^{(0)}(t) &= 1 \\
x^{(1)}(t) &= 1 + \int_{t_{init}}^{t} 2x^{(0)}(\tau)d\tau = 1 + 2(t - t_{init}) \\
x^{(2)}(t) &= 1 + \int_{t_{init}}^{t} 2x^{(1)}(\tau)d\tau = 1 + \int_{t_{init}}^{t} 2[1 + 2(\tau - t_{init})]d\tau \\
&= 1 + \int_{t_{init}}^{t} 2 + 4(\tau - t_{init})d\tau = 1 + 2(t - t_{init}) + 2(t - t_{init})^2 \\
x^{(3)}(t) &= 1 + \int_{t_{init}}^{t} 2x^{(2)}(\tau)d\tau = 1 + \int_{t_{init}}^{t} 2[1 + 2(\tau - t_{init}) + 2(\tau - t_{init})^2]d\tau \\
&= 1 + \int_{t_{init}}^{t} [2 + 4(\tau - t_{init}) + 4(\tau - t_{init})^2]d\tau \\
&= 1 + 2(t - t_{init}) + 2(t - t_{init})^2 + \tfrac{4}{3}(t - t_{init})^3
\end{aligned}$$

The solution to this EIVP is $x(t) = e^{2t}$, to which we can compute order n Taylor approximation at the point t_0 with

$$x(t) = \sum_{k=0}^{n} \frac{1}{k!} \frac{d^k x}{dt^k}(t_{init})(t - t_{init})^k$$

since $x(t) = e^{2t}$ we know that $\frac{d^k x}{dt^k}(t_{init}) = 2^k$.

We present the up to order 3 Taylor approximation and Picard iterations in the following table.

k	Picard iteration	Taylor approximation
0	1	2^0
1	$1 + 2(t - t_{init})$	$1 + \frac{2^1}{1!}(t - t_{init})$
2	$1 + 2(t - t_{init}) + 2(t - t_{init})^2$	$1 + \frac{2^1}{1!}(t - t_{init}) + \frac{2^2}{2!}(t - t_{init})^2$
3	$1 + 2(t - t_{init}) + 2(t - t_{init})^2 + \frac{4}{3}(t - t_{init})^3$	$1 + \frac{2^1}{1!}(t - t_{init}) + \frac{2^2}{2!}(t - t_{init})^2 + \frac{2^3}{3!}(t - t_{init})^3$

from which we can easily see that the corresponding Picard iteration and Taylor approximation are equal.

For convenience let us now assume that any EIVP from here onwards will have $t_{init} = 0$.

3.3.1 Computing Flowpipes

To restrict the memory requirements of the Taylor models involved we are going to use k^{th}-order Taylor models to represent the flow. Let us also assume the initial conditions are given by a Taylor model.

A Taylor model for the flow of the EIVP is computed in three phases.

1. Compute an approximation of the solution using k^{th}-order Picard iteration. This is done by applying the Picard operator k-times to the initial set. Further iterations[4] do not change this polynomial part $\vec{p}$ of this model. For efficiency, these computations can ignore the interval parts of Taylor models and discard monomials of order greater than k.

2. Guess an interval $\mathbf{i}$ with the hope that the Taylor model $(\vec{p}, \mathbf{i})$ encloses all the solutions to the EIVP that have start values at time t_{init} in the interval defined by the Taylor model $\vec{x}_0$.

   Compute $(\vec{p'}, \mathbf{i'}) = \mathbb{P}^I_{\vec{f}}((\vec{p}, \mathbf{i}))$ and derive interval enclosures of all monomials in $\vec{p'}$ of order greater than k in order to arrive at a k^{th} order Taylor model $(\vec{p}, \mathbf{i''})$ that encloses $(\vec{p'}, \mathbf{i'})$. Using the Banach fixed point theorem it can be shown that if $\mathbf{i''} \subseteq \mathbf{i}$, then the Taylor models $(\vec{p}, \mathbf{i})$ and $(\vec{p}, \mathbf{i''})$ both enclose all the EIVP solutions.

   If the inclusion test $\mathbf{i''} \subseteq \mathbf{i}$ fails, typically, another $\mathbf{i}$ is searched for by successively trying larger guesses. Alternatively, if finding a larger suitable $\mathbf{i}$ fails shorter time step sizes or higher-order Taylor models can also be used.

3. Tighter enclosures of the set of solutions can be found by iterating the Picard operator on $(\vec{p}, \mathbf{i''})$. In practice, these iterations are terminated when the decrease in the remainder interval $\mathbf{i}$ size on an iteration is less than some threshold.

Phases 2 and 3 modify result in Taylor models that differ only by the remainder interval. As observed in the PhD thesis of the Flow* implementer [Che15, p. 73], efficiency gains[5] can be made in the application of the Picard operator here by caching the polynomial bounds used to compute new intervals.

After the last phase, the resulting Taylor model is evaluated at the time step end time t_{final} to give a Taylor model that encloses the solution values at the time step end and that can be used as the initial condition for the next time step (since it is a valid enclosure of the solution set at that time).

We present the algorithms for computing flow for EIVP in Algorithm 2. The lines 1-4 correspond to the first phase, lines 5-10 correspond to the second phase and lines 11-15 correspond to the third phase.

[4] Given that the arithmetic involved in further iterations is Taylor model arithmetic of order k.

[5] Applying Picard operator can be order of magnitude times faster.

Algorithm 2: Computing flow for EIVP

input : EIVP $(\vec{f}, \vec{\mathcal{T}}^0(\vec{a}), [0, t_{final}]$

output: k^{th}-order Taylor model for flow $\vec{\mathcal{T}}$ of the EIVP

$\vec{x}^{(0)}(t,\vec{a}) \leftarrow \vec{\mathcal{T}}^0(\vec{a})$;
for $i \leftarrow 1$ **to** k **do**
  $\vec{x}^{(i)}(t,\vec{a}) \leftarrow \mathbb{P}^I_{\vec{f}}(\vec{x}^{(k)})(t,\vec{a})$;
end
$\mathbf{i} \leftarrow$ initialGuess;
$\vec{\mathcal{T}}(t,\vec{a}) \leftarrow (\vec{x}^{(i)}(t,\vec{a}), \mathbf{i})$;
while $\vec{\mathcal{T}}(t,\vec{a}) \not\subseteq \mathbb{P}^I_{\vec{f}}(\vec{\mathcal{T}})(t,\vec{a})$ **do**
  $\mathbf{i} \leftarrow$ enlarge($\mathbf{i}$);
  $\vec{\mathcal{T}}(t,\vec{a}) \leftarrow (\vec{x}^{(i)}(t,\vec{a}), \mathbf{i})$;
end
$(\vec{p}, \mathbf{i}'') \leftarrow \mathbb{P}^I_{\vec{f}}(\vec{\mathcal{T}})(t,\vec{a})$;
while $\frac{|W(\mathbf{i}'')|}{|W(\mathbf{i})|} < threshold$ **do**
  $\vec{\mathcal{T}}(t,\vec{a}) \leftarrow (\vec{x}^{(i)}(t,\vec{a}), \mathbf{i}'')$;
  $(\vec{p}, \mathbf{i}'') \leftarrow \mathbb{P}^I_{\vec{f}}(\vec{\mathcal{T}})(t,\vec{a})$;
end
return $\vec{\mathcal{T}}(t,\vec{a})$;

3.3.2 Initial Taylor Model

We view computing flowpipes for different time steps as separate EIVP problems. Besides the first one, the initial conditions in all of them are represented by Taylor models. On the first one, the initial conditions are often represented by either points or intervals. Let us briefly discuss converting them into Taylor models.

Point initial conditions can be viewed as Taylor models with a constant polynomial and no remainder, therefore, let us only consider intervals, i.e. let the initial conditions $\vec{X}_{init}$ be an interval box.

A Taylor model consists of the polynomial and the interval. Therefore, we have 2 options to convert intervals into Taylor models:

1. by setting the remainder intervals to be elements of $\vec{X}_{init}$ or
2. by parametrizing the initial condition interval.

Choosing option 1 will result in Taylor models that consist solely of interval remainder terms. Arithmetic operations will not change this property and we can conclude that using this approach the Taylor model based validated integration would degenerate to interval-based validated integration, for this reason, we are going to ignore option 1.

A simple way to achieve option 2 is to use the identity Taylor model with $\vec{X}_{init}$ as the domain D. Keeping in mind that Taylor models introduce the least amount of over-approximation when their domain is the unit box, we note that it is favourable to convert the proposed Taylor model to have domain as the unit box. This is always possible without introducing any over-approximation since the elements of the Taylor model proposed so far are independent of each other.

Example 8 (Creating Taylor model from interval box)**.** Let the initial conditions be

$$\vec{X}_{init} = \begin{bmatrix} [0,1] \\ [1,2] \\ [0,2] \end{bmatrix}$$

By introducing parameters a_1, a_2 and a_3, we can represent this set by the Taylor model

$$\vec{T} = \begin{bmatrix} a_1 + [0,0] \\ a_2 + [0,0] \\ a_3 + [0,0] \end{bmatrix}$$

where the parameters $\vec{a}$ take values from domain $\vec{X}_{init}$.

We can convert this Taylor model into a more favourable one by centring and scaling the domain. That is, we get

$$\vec{\mathcal{T}}' = \begin{bmatrix} \frac{1}{2} + \frac{1}{2}a_1 + [0,0] \\ \frac{3}{2} + \frac{1}{2}a_2 + [0,0] \\ 1 + a_3 + [0,0] \end{bmatrix}$$

where the parameters $\vec{a}$ take values from domain $[-1,1]^3$.

In the following, we assume that interval $\vec{X}_{init}$ is converted to a Taylor model unless specified otherwise.

3.4 Preprocessing Between Integration Steps

Taylor model arithmetic is built on top of interval arithmetic. Interval arithmetic often introduces unnecessary over-approximation due to the wrapping effect and dependency problem. This unnecessary over-approximation propagates to Taylor model arithmetic and therefore can result in worse flows in the Taylor model based validated integration.

The *wrapping effect* was first observed by Moore in [Moo65]. It can be briefly described by the need to enclose a region with an interval box. The wrapping effect manifests in dimension 2 or higher. A classical example of wrapping effect is rotating a box by 45 degrees and then wrapping it in the original coordinate system (see Figure 3.1). The over-approximation arising from wrapping effect scales linearly with the step size and therefore cannot be controlled by reducing it. Wrapping effect can be mitigated by using a moving coordinate system that is introducing less over-approximation [Moo65, Moo66, Eij81, Loh87, Loh88].

Interval methods are sometimes overly pessimistic in the bounds computed. The *dependency problem* is often the source of the overestimation. The dependency problem can manifest when the same interval appears multiple times in the calculations. The issue is that all the occurrences of the interval are treated independently i.e. it is lost that the variable (whose value is inside of that interval) should have the same fixed value in all the places it is occurring.

Example 9 (Dependency problem)**.** Suppose we want to compute the value of the interval-based function $f : \mathbb{IR} \to \mathbb{IR}$, where f is defined as $f(x) = x^2 - x$ with $x \in [0,1]$. The range of f for $x \in [0,1]$ is the interval $[-\frac{1}{4}, 0]$, but if we use interval arithmetic we

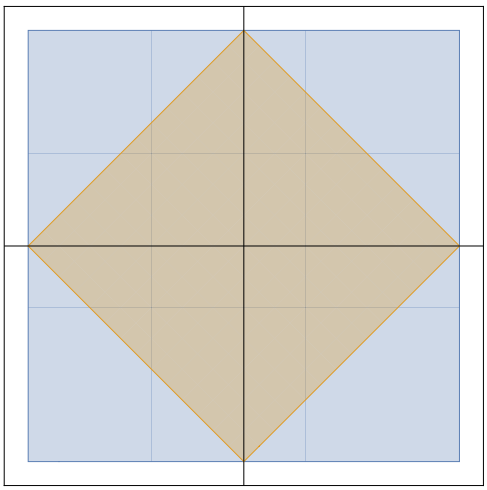


Figure 3.1: Enclosing a rectangle with and interval box after 45 degree rotation.

get

$$f([-1,1]) = [0,1]^2 - [0,1] = [-1,1].$$

The wrapping effect and the dependency problem both arise in Taylor model arithmetic from interval remainder terms. Both of them can be mitigated by reducing the width of the interval remainder term. We discuss the techniques shrink wrapping and preconditioning [MB11, NJN07] which do that.

3.4.1 Shrink Wrapping

Shrink wrapping is the first scheme developed by Berz and Makino [MB11] to try to reduce over-approximation from the remainder interval. In short, shrink wrapping tries to push the interval remainder term into the symbolic part of the Taylor model by modifying the symbolic part slightly. However, it has stability problems and may become inapplicable after repeated applications (between the integration of different time steps).

More precisely, between integration steps, the Taylor model based validated integration calculates a Taylor model $\vec{\mathcal{T}} = (\vec{p}, \vec{\mathbf{i}})$ to be used as an initial set for the next integration step. As said before, during the integration step the size of the interval remainder term $\vec{\mathbf{i}}$ affects the over-approximation introduced. Shrink wrapping reduces the over-approximation by wrapping $\vec{\mathcal{T}}$ with another Taylor model that has the zero interval for the remainder interval. Unfortunately, this process itself usually introduces significant over-approximation, the success of the method depends on the amount of the over-approximation. For practical reasons, shrink wrapping is not applied at every

integrations step, but instead when the size of the interval remainder term grows too big. The precise size of what is considered too big is defined by some heuristic.

Let us now discuss the method itself. Let us assume that the n-element Taylor model $\vec{\mathcal{T}}$ consists of constant part $\vec{c}$, linear part $C\vec{a}$, non-linear part $\vec{n}(\vec{a})$ and interval remainder $\vec{\mathbf{i}}$.

$$\vec{\mathcal{T}} = \vec{c} + C\vec{a} + \vec{n}(\vec{a}) + \vec{\mathbf{i}}$$

Let us also assume that the domain $\vec{\mathcal{T}}$ is the n-dimensional unit box D.

Shrink wrapping can be summarized by applying a series of transformations:

$$\begin{aligned}
&\vec{c} + C\vec{a} + \vec{n}(\vec{a}) + \vec{\mathbf{i}} \\
&\qquad\downarrow \quad \lambda\vec{z}.\vec{z} - \vec{c} && (3.2)\\
&C\vec{a} + \vec{n}(\vec{a}) + \vec{\mathbf{i}} \\
&\qquad\downarrow \quad C^{-1}\cdot && (3.3)\\
&C^{-1}(C\vec{a} + \vec{n}(\vec{a}) + \vec{\mathbf{i}}) = I\vec{a} + C^{-1}\vec{n}(\vec{a}) + C^{-1}\vec{\mathbf{i}} \\
&\qquad\cap \quad \lambda\vec{z}.q(\vec{z}\{\mathbf{i} \mapsto [0,0]\}) && (3.4)\\
&q(\vec{a} + C^{-1}\vec{n}(\vec{a})) \\
&\qquad\downarrow \quad C\cdot && (3.5)\\
&Cq(\vec{a} + C^{-1}\vec{n}(\vec{a})) = qC\vec{a} + q\vec{n}(\vec{a}) \\
&\qquad\downarrow \quad \lambda\vec{z}.\vec{z} + \vec{c} && (3.6)\\
&\vec{c} + q(C\vec{a} + \vec{n}(\vec{a}))
\end{aligned}$$

where I is the n-dimensional identity matrix and q is a scalar.

Transformation (3.2) removes the constant part and centres the set around the origin. (3.3) transforms the coordinate system such that the linear part describes a unit box. The core of the shrink wrapping lies in (3.4), where the polynomial part is inflated by multiplying with a scalar q. q is picked such that the inclusion

$$Range(\vec{a} + C^{-1}\vec{n}(\vec{a}) + C^{-1}\vec{\mathbf{i}}, D) \subseteq Range(q(\vec{a} + C^{-1}\vec{n}(\vec{a})), D) \tag{3.7}$$

is satisfied. If that is the case, then after applying reverse transformation (3.5) and (3.6), we also get the inclusion

$$\vec{c} + C\vec{a} + \vec{n}(\vec{a}) + \vec{\mathbf{i}} \subseteq \vec{c} + q(C\vec{a} + \vec{n}(\vec{a}))$$

giving us the desired Taylor model with $[0,0]^n$ as the interval remainder term.

To satisfy the inclusion (3.7), q is defined as

$$q = 1 + d\frac{1}{(1-(n-1)t)(1-s)}$$

where s,t satisfy the conditions:

$$\begin{aligned} 1 > s \geq |n'_x(\vec{a})| \quad & \forall \vec{a} \in D, \quad x \in \vec{x} \\ \tfrac{1}{n} > t \geq \left|\tfrac{\partial n'_x(\vec{a})}{\partial a}\right| \quad & \forall \vec{a} \in D, \quad x \in \vec{x}, a \in \vec{a} \end{aligned}$$

where n'_x is the element corresponding to x in $C^{-1}\vec{n}(\vec{a})$ and d is picked such that $C^{-1}i \subseteq d[-1,1]^n$

The factor q is made up of two parts, we will briefly describe both of them.

The part $d\frac{1}{1-(n-1)t}$ describes how much the addition of $\vec{\mathbf{i}}$ can extend the surface of the Taylor model in any direction.

The part $\frac{1}{1-s}$ is used to guarantee that the range of the Taylor model moves out by a required amount. More precisely, the box $(1-s)[-1,1]^n$ lies inside of the range of the Taylor model. Multiplying the Taylor model $\vec{T}$ with any factor $q > 1$ will guarantee that the borders of the box $(1-s)[-1,1]^n$ move out by the amount of $(1-s)(q-1)$ in all directions. Because this box lies entirely inside of the range of the Taylor model, this means that the border of the range of the Taylor model also moves out by at least the same amount.

Shrink wrapping may be inapplicable or applicable but leading to very large over-approximations. The reasons for that are the following:

1. the matrix C is ill-conditioned. If that is the case, then the matrix C is not invertible or invertible but leading to large overestimations,

2. s and t maybe become too large. Namely, shrink wrapping requires that $s < 1$ and $t < \frac{1}{n}$.

We will briefly describe ways to address these problems.

The blunting technique [MB11] is used to address the first issue. The blunting technique is used to better condition the matrix C. The blunting technique has a negative side effect that since it increases the interval remainder terms, it also increases the over-estimation.

Besides being related to the domain of the problem, the second issue can also arise when integration itself introduces a lot of over-approximation. Therefore, decreasing s or t by enclosing some terms of the polynomial part by the interval box and them moving them to the interval remainder will not have a noticeable effect, since interval remainder term already included a lot of over-approximation.

Note that either one of the two options will address one issue and amplify the other one.

3.4.2 Preconditioning

Preconditioning is the second scheme developed by Berz and Makino [MB11]. Compared to shrink wrapping preconditioning has a couple of advantages. First, some preconditioning methods are stable[6] and secondly, preconditioning tends to offer more precise results.

In the Taylor model based validated integration, the initial condition in EIVP is usually given by (or transformed into) a non-factored Taylor model. In naive methods (and in the shrink wrapping method), the flow for a time step is computed also as a non-factored Taylor model. To get the initial condition for the next time step, this flow (which includes the time parameter) is specified with a value for the time parameter resulting in another non-factored Taylor model. Preconditioning uses a similar process but uses factored Taylor models to represent the initial condition, flow and the initial condition for the next time step.

The merit of using factored Taylor models is that the initial conditions[7] can be replaced with different initial conditions that introduce less over-approximation[8]. We call such factored Taylor models *preconditioned Taylor models*.

We will discuss preconditioned Taylor models in more detail shortly, but first, let us mention that in integration only the left model is used. We will refer to [NJN07] for the proof of the following theorem.

Theorem 3.4.1. Let $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ be the factored Taylor model that encloses the flow of the ODE at time t_{init}. Let $\vec{\mathcal{U}}^*$ be the flow of the EIVP $(\vec{f}, \vec{\mathcal{U}}, [t_{init}, t_{final}])$, then

$$\vec{\mathcal{U}}^* \circ \vec{\mathcal{V}}$$

is the flow for of EIVP $(\vec{f}, \vec{\mathcal{U}} \circ \vec{\mathcal{V}}, [t_{init}, t_{final}])$

[6] We mean that the matrices involved in preconditioning will not become more and more ill-conditioned as the integration goes on.

[7] We remind the reader that the initial conditions do not include the time parameter. The time parameter is present of the flow during a time step but specified with a value in order to compute the initial set for the next time step.

[8] By initial conditions, we mean initial conditions for a single time step under consideration. The flow for each time step is computed by preconditioning the initial conditions for that particular time step and then using the result of the preconditioning in integration. In other words, applying preconditioning alternate with applying integration.

When using factored Taylor models, then the left model essentially acts as a variable substitution in integration. By moving terms form the left model to the right model we have an option to make the left model more suitable for integration.

To be precise, preconditioning is used to replace the flow $\vec{\mathcal{U}}^* \circ \vec{\mathcal{V}}$ at the end of the time step with a more favourable factorisation $\vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'$ such that the condition

$$\vec{\mathcal{U}}^* \circ \vec{\mathcal{V}} \subseteq \vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'$$

is satisfied. Let us note that preconditioning introduces over-approximation both on the right model and on the left model (which will be then amplified by integration). However, this over-approximation is outweighed by the effectiveness of the method. The effectiveness of the method comes mainly from the fact that the left models after being preconditioned has an empty remainder interval, making it possible to introduce less over-approximation during integration.

We will now present the preconditioning method itself. Let the left model in the factorised model $\vec{\mathcal{U}}^* \circ \vec{\mathcal{V}}$ that we wish to precondition be with the form

$$\vec{\mathcal{U}}^* = \vec{c} + \vec{C}^* \vec{a} + \vec{n}^*(\vec{a})$$

where $\vec{c}$ is the constant part[9], $\vec{C}^* \vec{a}$ is the linear part and $\vec{n}(\vec{a})$ is the non-linear part of $\vec{\mathcal{U}}^*$.

We can precondition this model with the following transformations:

$$\begin{aligned}
\vec{\mathcal{U}}^* \circ \vec{\mathcal{V}} &= [\lambda \vec{a}.\ \vec{c} + \vec{C}^* \vec{a} + \vec{n}^*(\vec{a})] \circ \vec{\mathcal{V}} \\
&= \vec{c} + [\lambda \vec{a}.\ \vec{C}^* \vec{a} + \vec{n}^*(\vec{a})] \circ \vec{\mathcal{V}} \\
&= \vec{c} + (\lambda \vec{a}.\ C\vec{a}) \circ \left((\lambda \vec{a}.\ C^{-1} \vec{a}) \circ [\lambda \vec{a}.\ C^* \vec{a} + \vec{n}^*(\vec{a})] \circ \vec{\mathcal{V}} \right) \\
&= \vec{c} + (\lambda \vec{a}.\ C\vec{a} + [0,0]^n) \circ \\
&\quad \left([(\lambda \vec{a}.\ C^{-1} \vec{a}) \circ (\lambda \vec{a}.\ C^* \vec{a}) + (\lambda \vec{a}.\ C^{-1} \vec{a}) \circ (\lambda \vec{a}.\ \vec{n}^*(\vec{a}))] \circ \vec{\mathcal{V}} \right) \\
&= (\lambda \vec{a}.\ \vec{c} + \vec{C}\vec{a} + [0,0]^n) \circ \\
&\quad \left([\lambda \vec{a}.\ \vec{C}^{-1} \vec{C}^* \vec{a} + \vec{C}^{-1} \vec{n}^*(\vec{a})] \circ \vec{\mathcal{V}} \right) \\
&= \vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'
\end{aligned} \tag{3.8}$$

where $\vec{C}$ is the desired linear and

$$\begin{aligned}
\vec{\mathcal{U}}' &= (\lambda \vec{a}.\ \vec{c} + \vec{C}\vec{a} + [0,0]^n) \\
\vec{\mathcal{V}}' &= [\lambda \vec{a}.\ \vec{C}^{-1} \vec{C}^* \vec{a} + \vec{C}^{-1} \vec{n}^*(\vec{a})] \circ \vec{\mathcal{V}}
\end{aligned}$$

[9] In order to make the presentation in the next chapter more convenient, we are viewing matrices as a vector of its rows .

are the left and right models after preconditioning.

We note that the left model does not include any interval remainder term after preconditioning. Overestimation in integration is therefore also mitigated since there will be less of a dependency problem and wrapping effect.

Let us now discuss the over-approximation coming from applying the preconditioning. The main source of this is the operator $\lambda\vec{a}.\,\vec{C}^{-1}\vec{C}^{*}\vec{a}+\vec{C}^{-1}\vec{n}^{*}(\vec{a})$. To be more precise, two aspects introduce overestimation:

a) the operation $\vec{C}^{-1}\vec{n}^{*}(\vec{a})$ involves multiplication of a matrix with interval remainder,

b) the need to limit the order of Taylor model arithmetic.

The overestimation coming from a) scales linearly with the condition number of C. For b), the composition used in computing the new right model likely results in some terms whose order exceeds the limit. These terms are bounded with an interval and added to the remainder interval of the right model.

It should be noted that the range of the preconditioned right model $\vec{\mathcal{V}}'$ is not the same as original right model $\vec{\mathcal{V}}$. Since we need to be able to specify the domain of preconditioned left model $\vec{\mathcal{U}}'$ during the next integration step we need to address this issue. We can either

1. compute the range of the preconditioned right model $\vec{\mathcal{V}}'$ and set that as the domain of the preconditioned left model $\vec{\mathcal{U}}'$ or

2. scale the range of right model $\vec{\mathcal{V}}'$ such that it will be the unit box.

For practical purposes, the second option is more appealing. For this purpose, a linear diagonal transformation on the right model and applying its reverse on the left model can be used

$$\vec{\mathcal{U}}'\circ\vec{\mathcal{V}}'=\vec{\mathcal{U}}'\circ(S\circ S^{-1})\circ\vec{\mathcal{V}}'=(\vec{\mathcal{U}}'\circ S)\circ(S^{-1}\circ\vec{\mathcal{V}}')$$

where S is picked such that $Range(S^{-1}\circ\vec{\mathcal{V}}',D)\subseteq[-1,1]^{n}$.

Therefore after scaling, we get the left model for the next step as $\vec{\mathcal{U}}''=\vec{\mathcal{U}}'\circ S$ and the right model for the next step as $\vec{\mathcal{V}}''=S^{-1}\circ\vec{\mathcal{V}}'$.

3.4.2.1 Factoring Initial Conditions

The initial conditions for a EIVP are not usually described with a factored Taylor models, therefore before we can discuss preconditioning a factored Taylor model, we need to discuss how to factor a single Taylor model.

Let us assume that the initial conditions to EIVP are represented with Taylor model $\vec{\mathcal{T}}$[10] with domain D. Our goal is to represent $\vec{\mathcal{T}}$ as a composition of the left and the right model such that $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ is a factorisation of $\vec{\mathcal{T}}$.

In the literature, the initial factorisation is achieved by using $\vec{\mathcal{T}}$ as $\vec{\mathcal{U}}$ and the identity Taylor model as $\vec{\mathcal{V}}$, the domain of $\vec{\mathcal{V}}$ is set to be D and domain of left model does not need to be specified (Algorithm 3) [MB11, NJN07].

Algorithm 3: Simple factoring of Taylor models

input : Taylor model $\vec{\mathcal{T}}$
output: $\vec{\mathcal{U}}$ and $\vec{\mathcal{V}}$ such that $\vec{\mathcal{T}} = \vec{\mathcal{U}} \circ \vec{\mathcal{V}}$

1 $\vec{\mathcal{U}} \leftarrow \vec{\mathcal{T}}$;
2 **foreach** $x \in \vec{x}$ **do**
3 $\quad \mathcal{V}_x \leftarrow pv(x)$
4 **end**

Example 10 (Factoring a Taylor model)**.** Let the initial conditions be given with a Taylor model $\vec{\mathcal{T}}$ (from Example 8)

$$\vec{\mathcal{T}} = \begin{bmatrix} \frac{1}{2} + \frac{1}{2}a_1 + [0,0] \\ \frac{3}{2} + \frac{1}{2}a_2 + [0,0] \\ 1 + a_3 + [0,0] \end{bmatrix} \bullet D$$

where $D = [-1,1]^3$.

Which we can factor with Algorithm 3 as

$$\vec{\mathcal{T}} = \overbrace{\begin{bmatrix} \frac{1}{2} + \frac{1}{2}a_1 + [0,0] \\ \frac{3}{2} + \frac{1}{2}a_2 + [0,0] \\ 1 + a_3 + [0,0] \end{bmatrix}}^{\vec{\mathcal{U}}} \circ \overbrace{\begin{bmatrix} b_1 + [0,0] \\ b_2 + [0,0] \\ b_3 + [0,0] \end{bmatrix}}^{\vec{\mathcal{V}}} \bullet D.$$

We note that Algorithm 3 has limitations. Namely, it can only be applied if the number of variables is the same as the number of parameters. If that is not the case,

[10] If that is not the case, then we first need to convert them into a Taylor model as we discussed above.

we cannot use $\vec{\mathcal{T}}$ in composition with identity Taylor model and neither can we use the domain D as the domain of identity Taylor model.

The restriction that the number of variables is the same as the number of parameters is a reasonable one. Most initial conditions for EIVPs are represented with interval boxes, converting them into Taylor model will only ever produce models with the same number of variables as there are parameters.

Despite this, we are going to look at the general case, where we do not pose any restrictions at all on the Taylor model initial conditions. We have two reasons for it:

1. it is more natural for our compositional setting,

2. it is a richer way to define an EIVP.

To elaborate on 1, we will show later that in our compositional setting, we do not couple variables and parameters. To elaborate on 2, we can use unrestricted Taylor models to represented richer sets for EIVPs[11].

To be able to handle unrestricted Taylor models as initial conditions we propose Algorithm 4 to factor Taylor models, the idea is that we use the identity Taylor model on the left model together with the constant part of the original Taylor model and we set the right model as the original model without constant part.

Algorithm 4: Factoring unrestricted Taylor model

input : Taylor model $\vec{\mathcal{T}}$

output: $\vec{\mathcal{U}}$ and $\vec{\mathcal{V}}$ such that $\vec{\mathcal{T}} = \vec{\mathcal{U}} \circ \vec{\mathcal{V}}$

$\vec{c} \leftarrow \text{const}(\vec{\mathcal{T}})$;
$n \leftarrow \text{vars}(\vec{\mathcal{T}})$;
foreach $x \in \vec{x}$ **do**
    $\mathcal{U}_x \leftarrow c_x + pv(x)$
end
$\vec{\mathcal{V}} \leftarrow \vec{\mathcal{T}} - \vec{c}$;

Example 11 (Using Algorithm 4 with unrestricted Taylor model)**.** Suppose we have a Taylor model $\mathcal{T}$ which has 1 variable and 2 parameters. W.l.o.g. let the constant part of $\mathcal{T}$ be c and everything else be $\mathcal{T}^*(a_1, a_2)$ i.e. $\mathcal{T} = c + \mathcal{T}^*(a_1, a_2)$.

[11] For example, we can have the initial condition of one variable be dependent on the initial condition of another variable.

Then applying Algorithm 4 yields

$$\mathcal{T} = \overbrace{(c + a_1)}^{\vec{\mathcal{U}}} \circ \overbrace{\mathcal{T}^*(b_1, b_2)}^{\vec{\mathcal{V}}}$$

which is a factoring of $\mathcal{T}$.

We note that Algorithm 4 introduces one unique parameter for each variable of the left model, guaranteeing that we can always apply the preconditioning on the result[12].

We also note that if the input is an interval box converted to Taylor model then Algorithm 4 agrees with Algorithm 3 up to the scaling of the range of right Taylor model. This difference would be lost after preconditioning since scaling involved in preconditioning would make the result of both algorithms effectively the same.

Example 12 (Comparision of Algorithm 3 and Algorithm 4)**.** Let us use Algorithm 4 to factor the Taylor model of Example 10.

$$\vec{\mathcal{T}} = \begin{bmatrix} \frac{1}{2} + \frac{1}{2}a_1 + [0,0] \\ \frac{3}{2} + \frac{1}{2}a_2 + [0,0] \\ 1 + a_3 + [0,0] \end{bmatrix} \bullet D$$

Using Algorithm 4 we get

$$\vec{\mathcal{T}} = \overbrace{\begin{bmatrix} \frac{1}{2} + a_1 + [0,0] \\ \frac{3}{2} + a_2 + [0,0] \\ 1 + a_3 + [0,0] \end{bmatrix}}^{\vec{\mathcal{U}}} \circ \overbrace{\begin{bmatrix} \frac{1}{2}b_1 + [0,0] \\ \frac{1}{2}b_2 + [0,0] \\ b_3 + [0,0] \end{bmatrix}}^{\vec{\mathcal{V}}} \bullet D$$

after scaling the range of $\vec{\mathcal{U}}'$ we get

$$\vec{\mathcal{T}} = \overbrace{\begin{bmatrix} \frac{1}{2} + \frac{1}{2}a_1 + [0,0] \\ \frac{3}{2} + \frac{1}{2}a_2 + [0,0] \\ 1 + a_3 + [0,0] \end{bmatrix}}^{\vec{\mathcal{U}}} \circ \overbrace{\begin{bmatrix} b_1 + [0,0] \\ b_2 + [0,0] \\ b_3 + [0,0] \end{bmatrix}}^{\vec{\mathcal{V}}} \bullet D$$

which coincides with the factoring in Example 10.

In the following we will use Algorithm 4 to factor Taylor models.

[12] Due to the matrix of coefficients of the linear part of the left model being invertible.

3.4.2.2 Preconditioning Types

Preconditioning methods differ by what they consider to be the desired linear part in the preconditioned left model. To simplify the presentation, let us express that desired linear part as $\vec{C}\vec{a}$. Let us also note that it is computed from the linear part of the Taylor model $\vec{\mathcal{U}}$.

We define three preconditioning methods that we will later experiment with and refer to [MB11] for more in detail discussion of these and alternatives.

Definition 3.4.1 (Identity preconditioning)**.** *Identity preconditioning* sets C to be identity matrix, i.e. $\vec{C} = I$

Definition 3.4.2 (Parallelepiped preconditioning)**.** *Parallelepiped preconditioning* sets the linear part of preconditioned model to be the same as the linear part of the model to be preconditioned i.e. $\vec{C} = \vec{C}^*$.

Definition 3.4.3 (QR preconditioning)**.** *QR preconditioning* sets $\vec{C}$ to be the matrix Q from the QR decomposition of the matrix $\vec{C}^*$ where columns are sorted in descending order.

Let us briefly discuss the effect of different preconditioning methods.

Identity preconditioning moves the remainder interval to the right model. The symbolic part of the Taylor model is essentially substituted with a variable, minimising the Taylor model arithmetic operations during integration.

Parallelepiped preconditioning accumulates the non-linear and remainder terms in the right model while letting the constant part and linear part be in the left model. It is known that often in practical systems the matrix $\vec{C} = \vec{C}^*$ becomes more and more ill-conditioned as the integration time gets longer. As explained before this will lead to bigger overestimation.

In QR preconditioning, the elements of $\vec{C}$ are often close to 1 which means that polynomials in Taylor models are subject to smaller coordinate transformation, leading to smaller round-off errors. Unlike, parallelepiped preconditioning the matrix $\vec{C}$ is guaranteed to be orthogonal and therefore well-conditioned.

Chapter 4

Compositional Validated Integration

In this chapter, we view the Taylor model based validated integration in the compositional framework. We discuss the compositional nature often found in biological systems and the form it imposes on the ODEs describing such systems. We then explain the effects of this form on the integration process, the Picard operator and the Picard iteration.

We discuss the Picard operator in terms of which initial conditions are affecting it when it is applied to a single variable. We use this discussion to present the compositional naive Taylor model based validated integration method. In that presentation, we make it clear which parameters are affecting the flowpipe of a single variable. We conclude by presenting algorithms for computing the Picard operator, Picard iteration and integration used in the compositional naive method.

4.1 Introduction

Complicated systems are commonly built from smaller systems. This is often also the case in synthetic biological systems where usually smaller systems are engineered to compute a specific function and bigger systems are built using these smaller systems (for example logic gates in [Mic15b]). Often these smaller systems are independent of large parts of the bigger systems. We are going to argue that in cases like that it is possible to analyse these smaller subsystems separately and use the result of that to analyse the bigger systems.

To our best knowledge, none of the validated integration tools are using this to a degree that we are interested in[1]. We are going to argue that in some systems this

[1] See Related Works and Tools in Chapter 1 about the closest approaches.

approach could yield non-trivial performance gains. The gains stem from the fact that the validated integration algorithms depend heavily on the dimensions of the ODE system. This dependency on dimensions is further amplified when using the Taylor model based validated integration. We note that when using Taylor models, there is a secondary dependence on the number of parameters used in representing the initial conditions[2].

The core idea behind composition is that we partition the system into a set of systems called *components* whose solving is more efficient due to them having smaller dimensions.

There is a loose analogy with our approach and 'cone of influence' reduction in model checking. That is, when one is model checking a particular property, one can cut down the model checked system to just those parts that can influence the truth of the property. Likewise, when we compute flow for a particular variable, we can cut down the system to only those variables and parameters that influence it.

4.2 Compositional View of a System

Suppose that we have an EIVP (3.1) where the initial condition $\vec{X}_{init}$ is given by the Taylor model $\vec{T}$ with parameters $\vec{a}$ and domain D that we wish to view in the compositional setting.

In systems that we discussed in Section 4.1, it is often the case that the derivative of a variable does not depend on all other variables. To make a variable's effect on other variables more concrete we are going to use a dependency graph.

Definition 4.2.1 (Dependency graph)**.** Given an EIVP, we define the *dependency graph* $G = (\vec{x}, \rightarrow)$ as the graph with the variables $\vec{x}$ as vertices and an edge $y \rightarrow x$ whenever the ODE vector field $\vec{f}$ makes $\frac{dx}{dt}$ directly depend on y.

If the derivative for x is $\frac{dx}{dt} = f_x(y_1, \ldots, y_m)$ (with $\{y_1, \ldots, y_m\} \subseteq \vec{x}$), then the dependency graph includes an edge $y_i \rightarrow x$ for each y_i. We will call these variables y_is the immediate influencers of x.

Definition 4.2.2 (Immediate influencer)**.** Given an ODE system with the dependency graph $G = (\vec{x}, \rightarrow)$ we say that y is an *immediate influencer* of x if $y \rightarrow x$.

We denote the set of *all immediate influencers* of variable x as $\vec{y}_x$ and define it as

$$\vec{y}_x = \{y \mid y \rightarrow x,\ y \in \vec{x}\}$$

[2] Often each dimension introduces a parameter in the Taylor model.

We reserve the symbol y and $\vec{y}$ exclusively for immediate influencers. In the vector $\vec{y}$ the variables are ordered as they are ordered in the vector $\vec{x}$ of all system variables.

This dependency graph tells us that $\frac{dx}{dt}$ at some time t is influenced by the value of variable $z \in \vec{x}$ at times $t' \leq t$. Specifically, $\frac{dx}{dt}$ can be influenced by z just when $z \to^* x$, where $\to^*$ is the reflexive transitive closure of $\to$.

Definition 4.2.3 (Influencer)**.** Given an ODE system with dependency graph $G = (\vec{x}, \to)$ we say that z is an *influencer* of x if $z \to^* x$.

We call denote the set of *all influencers* of variable x as $\vec{z}_x$ and define it as

$$\vec{z}_x = \{z \mid z \to^* x,\ z \in \vec{x}\}$$

We reserve the symbol z and $\vec{z}$ exclusively for influencers. In the vector $\vec{z}$ the variables are ordered as they are ordered in the vector $\vec{x}$ of all system variables.

If $z \to^* x$ does not hold for some $x, z \in \vec{x}$, then the time evolution of x is independent of z. On the other hand, if we have both $z \to^* x$ and $x \to^* z$, then the evolution of variables x and z are mutually dependent.

In the following we will show that if

- $z \not\to^* x$ then variable z can be solved without solving the variable x,
- $z \to^* x$ and $x \to^* z$ then variables x and z have to solved at the same time.

This motivates us to view the sets of variables that need to be advanced together. We can find these sets by finding the strongly-connected components of G. These sets are central to our compositional view and we will call them components. In addition to the variables in components, we will make explicit what variables appear in the derivatives of these variables and the number of parameters in the Taylor models.

Definition 4.2.4 (Component)**.** Given a system EIVP (3.1) and its dependency graph G, we define a *component* as a strongly connected component of G. For component A

1. we denote the variables present there as $\vec{x}_A$,
2. we define the set of immediate influencers as

$$\vec{y}_A = \{y \mid y \to x \wedge x \in \vec{x}_A \wedge y \notin \vec{x}_A\}.$$

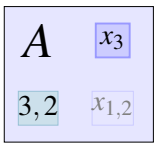


Figure 4.1: Graphical representation of component.

We leave out the variables present in the component out of its immediate influencers. We do this to make it easier to distinguish the variables for which we have already have computed the enclosures when solving a component.

When presenting a compositional system, we describe a component graphically as shown in Figure 4.1, the top left corner depicts the name of component ('A'), bottom left corner the number of parameters in the Taylor models (left and right model separated by commas, '$3,2$'), the bottom right corner[3] the immediate influencers of the component ('$x_{1,2}$') and the top right corner the variables of the component('x_3').

Observation 4.2.1. Components form a partition of the set of all variables.

Like we did with variables, we can illustrate how components influence each other with the dependency graph for components.

Definition 4.2.5 (Dependency graph for components)**.** Given an ODE system with the dependency graph $G = (\vec{x}, \rightarrow)$ and components $\vec{A}$. We define the dependency graph for components $G_C = (\vec{A}, \rightarrow_C)$ as a graph where nodes are components $\vec{A}$ and there is an edge from component A_1 to component A_2 if for some $x_1 \in \vec{x}_{A_1}$, $x_2 \in \vec{x}_{A_2}$ and we have that $x_1 \rightarrow x_2$.

Corollary 4.2.1. The dependency graph for components is acyclic.

The dependency graph for components tells us that if $A_1 \rightarrow_C^* A_2$ then at time t the value of a variable in component A_1 is influenced by a variable in component A_2 at time $t' \leq t$. Conversely, if $A_1 \not\rightarrow_C^* A_2$ then A_1 does not affect A_2 at all and we could solve A_2 while ignoring A_1 completely. This observation motivates our compositional approach.

Any topological sort of the component dependency graph defines a suitable ordering reflecting how components influence each other. In the following, we are going to argue that when using this sort to compute flow for components, we do not need to consider variables appearing in the components further down the order (and neither

[3]To save space in this graphical representation, we will appended indexes to denote multiple variables, i.e. $x_{1,2}$ denotes the set $\{x_1, x_2\}$. Similarly, $\{x_{1-3}\}$ denotes the set $\{x_1, x_2, x_3\}$.

their parameters). This makes it possible to view individual components as lower-dimensional systems (w.r.t. both the number of variables and the number of parameters).

We are also arguing that when we compute flows for any component, then we only need the flows of the components that are preceding it in that ordering. In order to specify which components we need precisely, we are going to define immediate component influencers.

Definition 4.2.6 (Immediate component influencer)**.** Given an ODE system with the component dependency graph $G_A = (\vec{A}, \rightarrow_C)$. We say that $A_1 \in \vec{A}$ is an *immediate component influencer* of $A_2 \in \vec{A}$ if $A_1 \rightarrow_C A_2$.

Example 13 (Compositional View)**.** Let us view the system described by the following reactions

$$\begin{aligned} r_1 &: \quad X_1 \xrightarrow{k_{r_1}^{\rightarrow}} X_2 + X_3 \\ r_2 &: \quad X_2 \underset{k_{r_2}^{\leftarrow}}{\overset{k_{r_2}^{\rightarrow}}{\rightleftharpoons}} X_3 \\ r_3 &: \quad X_2 \xrightarrow{k_{r_3}^{\rightarrow}} X_4 \end{aligned}$$

in the compositional setting.

We can model the dynamics of this system with EIVP where ODEs are

$$\begin{aligned} \frac{dx_1}{dt} &= -k_{r_1}^{\rightarrow} x_1 \\ \frac{dx_2}{dt} &= k_{r_1}^{\rightarrow} x_1 - (k_{r_2}^{\rightarrow} + k_{r_3}^{\rightarrow}) x_2 + k_{r_2}^{\leftarrow} x_3 \\ \frac{dx_3}{dt} &= k_{r_1}^{\rightarrow} x_1 + k_{r_2}^{\rightarrow} x_2 - k_{r_2}^{\leftarrow} x_3 \\ \frac{dx_4}{dt} &= k_{r_3}^{\rightarrow} x_2. \end{aligned}$$

In this system, the variables x_1, x_2, x_3 and x_4 model the concentrations of the chemicals X_1, X_2, X_3 and X_4, respectively. Let us also assume that the initial conditions are given by the Taylor model $\vec{\mathcal{T}}$

$$\vec{\mathcal{T}} = \begin{bmatrix} \mathcal{T}_{x_1}(a_1) \\ \mathcal{T}_{x_2}(a_2) \\ \mathcal{T}_{x_3}(a_3) \\ \mathcal{T}_{x_4}(a_4) \end{bmatrix} \bullet D$$

where D is the domain for parameters a_1, a_2, a_3 and a_4.

We give the dependency graph for this system in Figure 4.2 and the component dependency graph in Figure 4.3. We can see that we can the component A_1 (containing the variable x_1) is independent of everything else. Component A_2 (containing the

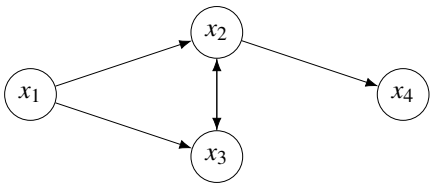


Figure 4.2: Dependency graph for example system.

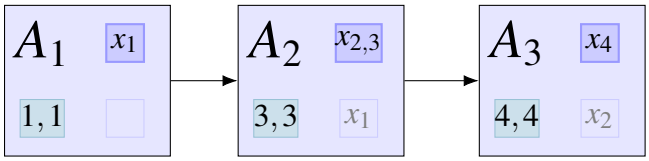


Figure 4.3: Component dependency graph for the example system.

variables x_2 and x_3) is influenced by the variable x_1 (from component A_1). Component A_3 (containing the variable x_4) is influenced by the variable x_2 (from component A_2).

4.3 Compositional Picard Operator

We intend to view a part of the system we want to find flowpipe for as independent of the rest of the system. In the Taylor model based validated integration we intend to do that by arguing that some of the parameters will never appear during the integration process for some variables.

We remind the reader that the integration in the validated integration consists of 3 phases. In phase 1, the polynomial parts of the approximation to the solution are found. In phase 2, the remainder interval box is found that ensures uniqueness and existence of the ODE solution within the Taylor model flowpipe. In phase 3, the remainder interval box is tightened while still ensuring uniqueness and existence. At the core of all these 3 phases is the repeated application of the Picard operator $\mathbb{P}^I_{\vec{f}}$.

The only thing introducing parameters in the Taylor models for variables in the integration process is the Picard operator. Therefore if we can guarantee that a repeated application of the Picard operator can only introduce a subset of the parameters for a specific variable then that property also holds for the whole integration process.

4.3.1 Picard Operator with Symbolic Initial Conditions

In phase 1 of the Taylor model based validated integration, the Picard operator can be viewed as a way to compute a more precise approximation of the flow given some ap-

proximation of the flow. This process involves operations with the values of the initial conditions. If initial conditions have parameters $\vec{a}$ then the approximations computed in phase 1 also have parameters $\vec{a}$.

In our presentation of the compositional approach we temporary abstract away from the parameters appearing in the initial conditions and instead we will use new parameters referring to the initial conditions themselves. We do this to decouple initial conditions from Taylor models representing them[4].

Definition 4.3.1 (Symbolic initial condition parameter)**.** We call x_i^0 the *symbolic initial condition parameter* or *symbolic parameter* referring to the initial conditions for the variable x_i. We use $\vec{x}^0$ to denote the vector of all such symbolic parameters.

We will use these symbolic initial condition parameters when iterating the Picard operator.

Lemma 4.3.1. When using symbolic initial condition parameters $\vec{x}^0$ in the Picard iteration, the Taylor models computed will have parameters t and $\vec{x}^0$.

Proof. We prove this lemma by induction on the number of applications of the Picard operator k.

Base case $k = 0$: The 0^{th} iteration sets the approximation for variable x_i to be the the initial condition of x_i

$$x_i^{(0)} = x_i^0$$

proving the base case trivially.

Inductive hypothesis: Suppose the lemma holds for some k^{th} iteration, $k \geq 0$.

Inductive step: By definition the $(k+1)^{th}$ Picard approximation is

$$x_i^{(k+1)} = x_i^0 + \int_0^t f_i(y_1^{(k)}(\tau,\vec{x}^0),\ldots,y_m^{(k)}(\tau,\vec{x}^0))d\tau$$

Since integration with respect to time will not introduce any new parameters[5] and neither does the addition of x_i^0, we can, therefore, conclude that the needed property holds for $x_i^{(k+1)}$. □

[4]This decoupling might seem superfluous at this point. To hint at things to come, we intend to use theorems presented here also in the section related to the preconditioned method and in there the parameters reference elements of the right model.

[5]Besides the time variable t.

It is easy to see that by substituting these new parameters with the Taylor models representing their initial conditions we can go back to the representation involving the parameters appearing in the initial condition $\vec{a}$.

Example 14 (Symbolic vs system parameters)**.** Suppose that our EIVP (3.1) has the ODEs

$$\begin{aligned} \frac{dx_1}{dt} &= x_1 \\ \frac{dx_2}{dt} &= x_1 + x_2 \end{aligned}$$

and initial conditions

$$\begin{aligned} x_1(0) &= 1 + a_1 \\ x_2(0) &= a_2 + a_3 \end{aligned}$$

Using symbolic parameters x_1^0 and x_2^0 we can initialize the Picard iteration as

$$\begin{aligned} x_1^{(0)}(t,\vec{x}^0) &= x_1^0 \\ x_2^{(0)}(t,\vec{x}^0) &= x_2^0 \end{aligned}$$

from which we can continue with

$$\begin{aligned} x_1^{(1)}(t,\vec{x}^0) &= x_1^0 + \int_0^t x_1^0 d\tau = x_1^0 + x_1^0 t \\ x_2^{(1)}(t,\vec{x}^0) &= x_2^0 + \int_0^t x_1^0 + x_2^0 d\tau = x_2^0 + (x_1^0 + x_2^0)t \end{aligned}$$

as the first Picard iteration.

Alternatively, using the values $x_1(0)$ and $x_2(0)$ we can initialize the Picard iteration as

$$\begin{aligned} x_1^{(0)}(t,\vec{a}) &= 1 + a_1 \\ x_2^{(0)}(t,\vec{a}) &= a_2 + a_3 \end{aligned}$$

From which we can continue with

$$\begin{aligned} x_1^{(1)}(t,\vec{a}) &= 1 + a_1 + \int_0^t 1 + a_1 d\tau = 1 + a_1 + (1 + a_1)t \\ x_2^{(1)}(t,\vec{a}) &= a_2 + a_3 + \int_0^t 1 + a_1 + a_2 + a_3 d\tau = a_2 + a_3 + (1 + a_1 + a_2 + a_3)t \end{aligned}$$

as the first Picard iteration.

We can observe that we substitute in the values of $x_1(0)$ and $x_2(0)$ for x_1^0 and x_2^0, respectively, we get the same results for both iterations.

Now that we have symbolic initial condition parameters, let us also relate them to the influencers.

Definition 4.3.2 (Symbolic initial condition parameters of the influencers)**.** Given an ODE system with dependency graph $G = (\vec{x}, \rightarrow)$ and symbolic initial conditions parameters $\vec{x}^0$ we define the initial conditions of the influencers for the variable x as

$$\vec{z}_x^0 = \{z^0 \mid z \in \vec{z}_x\}.$$

Using symbolic initial condition parameters of the influencers we can restrict the parameters in the Taylor models computed as the result of the Picard iteration.

Lemma 4.3.2 (Symbolic initial condition parameter invariance in the Picard iteration)**.** Suppose that we have an EIVP (3.1) with symbolic initial conditions $\vec{x}^0$, then the Picard iteration for variable x will only depend on parameters t and $\vec{z}_x^0$.

Proof. Let us also assume that explicit variable dependencies for x_i are $y_1, \ldots, y_m$, i.e.

$$\frac{dx_i}{dt} = f(y_1, \ldots, y_m).$$

We prove the lemma by induction on the number of application of Picard operator k.

Base case $k = 0$: The zeroth application sets the Picard iteration as

$$x_i^{(0)}(t, \vec{x}^0) = x_i^0$$

by definition $x_i \in \vec{z}_{x_i}$ and it follows that $x_i^0 \in \vec{z}_{x_i}^0$ proving the property for the base case.

Inductive hypothesis: Suppose the lemma holds for some k^{th} step, $k \geq 0$.

Inductive step: The $(k+1)^{th}$ Taylor polynomial for the variable x_i can be found with the expression

$$x_i^{(k+1)}(t, \vec{x}^0) = x_i^0 + \int_0^t f_i(y_1^{(k)}(\tau, \vec{x}^0), \ldots, y_m^{(k)}(\tau, \vec{x}^0))d\tau \tag{4.1}$$

where y_js are the immediate influencers of x_i.

We have already addressed the term x_i^0 in the base case, so we now only need to analyse the integral. The expression under the integral is

$$f_i(y_1^{(k)}(\tau, \vec{x}^0), \ldots, y_m^{(k)}(\tau, \vec{x}^0))$$

we can see that is expression is built exclusively from the k^{th} order Picard iterations for x_is immediate influencers. Let us analyse these variables in more detail.

$y_j^{(k)}(\tau, \vec{x}^0)$ is a k^{th} order Picard approximation, meaning that we can apply the induction hypothesis and express it in terms of initial conditions of the variable y_j influencers. That is, by applying the induction hypothesis we can write

$$y_j^{(k)}(\tau, \vec{x}^0) = y_j^{(k)}(\tau, \vec{z}_{y_j}^0).$$

Let w^0 be a symbolic initial condition parameter present in $\vec{z}_{y_j}^0$. From the induction hypothesis, we then can conclude that the variable w is an influencer of y_j, i.e. $w \to^* y_j$.

Since $y_j \to x$ we can conclude that $w \to^* x$, this means that $w^0 \in \vec{z}_x^0$. Since both w^0 and y_j were unrestricted we can conclude that any parameter present in the expression under the integral is in the set $\vec{z}_x^0$. Integrating this expression will not introduce any parameters[6], therefore we can conclude that lemma holds. □

Theorem 4.3.3 (Symbolic initial condition parameter invariance of the flow)**.** The flow for the EIVP using symbolic initial condition parameters will only have parameters t and $\vec{z}_x^0$ present in the element corresponding to variable x.

Proof. Computing the flow for EIVP consists of 3 phases:

1. finding the polynomial part of the enclosure,
2. validating the existence and uniqueness of the solution,
3. refining the flow to contain less over-approximation.

All of these phases are applying Picard operator and nothing else. The two differences with the application of the Picard operator discussed so far how are the Picard iterations initialized in phases 2 and 3 and the operations involving the interval handled.

In short, the result of phase 1 is the initial value of the iteration in phase 2 and the result of phase 2 is the input of phase 3. Therefore, in this aspect, we still look at this as simply a longer Picard iteration.

We can also note that operations involving the remainder term cannot introduce any parameters.

Therefore, we can conclude that simply applying Lemma 4.3.2 proves this theorem. □

4.4 Compositional Naive Taylor Model Method

In the previous section, we have presented the result of the Picard iteration as a Taylor model where parameters reference the initial conditions symbolically. We also showed how only some of the initial conditions can be present there for specific variables. In this section, we are going to look at naive Taylor model method and argue that we can restrict the parameters appearing in the initial conditions (non-symbolically).

To begin with, we need to define the parameters (of the initial conditions) that are relevant to a variable, i.e. the parameters that are present either in the initial condition of the variable or in the initial condition of any of its influencers.

[6] Besides possibly the time parameter t

Definition 4.4.1 (Parameters of the influencers)**.** In EIVPs where the initial condition is given by a Taylor model, we define the *parameters of the influencers* of the variable x as

$$\vec{a}_x = \bigcup_{z \in \vec{z}_x} ip(z)$$

The Taylor model computed for variable x in the Picard iteration will only include parameters $\vec{a}_x$.

Theorem 4.4.1 (Parameter invariance of the Picard iteration in the naive Taylor model based validated integration)**.** Let us assume that we have an EIVP (3.1) where the initial conditions have parameters $\vec{a}$ over domain D. Then the Picard iterations for variable x depends only on parameters $\vec{a}_x$.

Proof. From Lemma 4.3.2 we know that the Picard iteration for variable x can be expressed in terms of parameter t and symbolic initial condition parameters $\vec{z}_x^0$. From the definition of $\vec{z}_x^0$ we know that this means that if initial condition parameter z^0 is in the set $\vec{z}_x^0$ then z is an influencer of x i.e. $z \to^* x$. After substituting z^0 with $z(0)$ we introduce parameters $ip(z)$ in the resulting Taylor model. Keeping in mind that $z \to^* x$ and the definition of $\vec{a}_x$ we know that $ip(z) \subseteq \vec{a}_x$. Since variable z was unspecified can conclude that the theorem holds. □

In the integration procedure, we intend to focus on components (which essentially are sets of variables). Suppose we have an EIVP and partially ordered set of components $(\vec{A}, \preceq)$ and a sorting on components $\vec{A}$. We will describe how we can compute flowpipes for the components. We will assume that we compute flowpipes for components w.r.t. the sorting.

Let A be a component $A \in \vec{A}$. We remind the reader that $\vec{x}_A$ is the set of variables in A and $\vec{y}_A$ is the set of variables not in A but that are the immediate influencers of one or more variables in A. Our goal is to compute a flowpipe for component A. W.l.o.g. we can assume that we have flowpipes for all the components that are preceding component A.

As discussed above when computing a flowpipe for a variable we need to use the Picard iteration. We are now going to show how to define the Picard iteration for x as an operator with arguments $(t, \vec{a}_x)$ as opposed to an operator with arguments $(t, \vec{a})$.

We remind the reader that in the general case the j^{th} Picard iteration for variable $x \in \vec{x}_A$ is an operator of $(t, \vec{a})$

$$x^{(j)}(t,\vec{a}) = x^0(\vec{a}) + \int_0^t f_x(y_1^{(j-1)}(\tau,\vec{a}),\ldots,y_m^{(j-1)}(\tau,\vec{a}))d\tau$$

with $x^{(0)}(t,\vec{a}) = x^0(\vec{a})$.

Let us make a couple of observations:

1. From Theorem 4.4.1 we know that for a variable x the Picard iterations will only depend on parameters $(t,\vec{a}_x)$, i.e. we are justified as viewing the Picard iterations for x as being operators with just arguments $(t,\vec{a}_x)$ rather than $(t,\vec{a})$. The same reasoning applies also to the initial condition of x i.e. it is an operator with just arguments $\vec{a}_x$ rather than $\vec{a}$. We will use the domain restriction operator to restricting the operator $x^0(\vec{a})$ to be an operator on $\vec{a}_x$, we denote this as $x_0|_{\vec{a}_x}(\vec{a}_x)$. Similarly, we also know that the immediate influencer y_i of x can be viewed as operators on $(t,\vec{a}_{y_i})$ and since $\vec{a}_{y_i} \subseteq \vec{a}_x$ it is possible to view them as operators on $(t,\vec{a}_x)$. We will use the domain extension operator to extend the domain of operators for immediate influencer from $(t,\vec{a}_{y_i})$ to $(t,\vec{a}_x)$. We will denote the domain extension operator as $|^{t,\vec{a}_x}$.

2. All of the variables in a component are in the same strongly connected component of the dependency graph, they depend on the same set of parameters. Let us call this set of parameters as $\vec{a}_A$ i.e. $\vec{a}_x = \vec{a}_A$ for all $x \in \vec{x}_A$.

Putting 1 and 2 together we can rewrite the application of the Picard operator in the Picard iteration for the variable x (that is in component A) as

$$x^{(j)}(t,\vec{a}_A) = x^0|_{\vec{a}_A}|^{t,\vec{a}_A}(t,\vec{a}_A) + \int_0^t f_x(y_1^{(j-1)}|^{t,\vec{a}_A}(t,\vec{a}_A),\ldots,y_m^{(j-1)}|^{t,\vec{a}_A}(t,\vec{a}_A)d\tau. \quad (4.2)$$

where $x^{(0)}(t,\vec{a}_A) = x^0|_{\vec{a}_A}|^{t,\vec{a}_A}(t,\vec{a}_A)$.

Let us note that on the right side of this equation we need the initial condition of x as well as the $(j-1)^{th}$ Picard iterations of the immediate influencers of x. The initial condition is given in the EIVP, the immediate influencers of x have to be either in the same component A or in one of the preceding components. If they are in the same component then we can assume that we have them from computing the previous Picard iteration. If they are in the preceding components then we can assume that we have computed Picard iterations for them when we were solving those components.

Since integration only involves applying the Picard operator, we can note that computing flows for component A can be viewed as solving the EIVP which has variables $\vec{x}_A \cup \vec{y}_A$ and where we only need to solve variables $\vec{x}_A$ (since we already have flows

for variables $\vec{y}_A$). The initial conditions in this system may only contain parameters $\vec{a}_A$, the same holds for the initial conditions of its influencers. Therefore, instead of solving a system with variables $\vec{x}$ and parameters $\vec{a}$, we need to solve a (possibly) lower-dimensional system with variables $\vec{x}_A$ and parameters $\vec{a}_A$.

For the naive method, we give the pseudo-code of the compositional Picard operator in Algorithm 5, the pseudo-code for the compositional Picard iteration in Algorithm 6 and the pseudo-code the compositional integration in Algorithm 7.

All of these algorithms assume there exists a mapping from a variable to its component `CompLookup`, i.e. if the variable x is in the component A then `CompLookup[x]` returns A.

Algorithm 5 assumes that there exists a function called `LimitOrder` with Taylor model and positive integer k as argument which computes a Taylor model where all of the terms with order greater than k are pushed to the remainder interval[7]. The parameter `InflLookup` is used to find the Taylor model corresponding to the immediate influencers of the component A. When computing Picard iteration `InflLookup` is referring to the previous Picard iteration values. When validating the uniqueness and existence of the solution, then `InfLookup` is referring to the candidate enclosure. Note that the Taylor model corresponding to variables that are not in A need to be extended to include appropriate parameters.

For Algorithm 6 and Algorithm 7 components are augmented with fields `iter` and `flow` where the Picard iterations and flow are stored respectively.

Algorithm 7 assumes that there exists a constant `initialGuess` corresponding to the first remainder term used in guaranteeing the existence and uniqueness of the solution, a constant `threshold` which denotes when to stop the refinement of the flow and a function `enlarge` which takes an interval as an argument and returns a larger interval.

For compositional integration, we give the pseudo-code for integration over multiple time steps in Algorithm 8.

Example 15 (Parameter dependencies in a a compositional system)**.** We illustrate the parameter dependency in a compositional non-preconditioned integration by applying Picard operator to the EIVP with differential equations

$$\begin{aligned} \frac{dx_1}{dt} &= f_{x_1}(x_1) \\ \frac{dx_2}{dt} &= f_{x_2}(x_1, x_2) \\ \frac{dx_3}{dt} &= f_{x_3}(x_2, x_3) \end{aligned}$$

[7] Terms from polynomial part can be pushed to the remainder by first enclosing them with an interval box and then adding that box to the remainder interval.

Algorithm 5: Picard iteration step for component in naive method

input : component A, initial conditions $\vec{\mathcal{T}}^0$ for all variables, order k, mapping from variables $\vec{x}_A \cup \vec{y}_A$ to their Taylor models InflLookup.

output : application of the Picard operator with k-order Taylor model arithmetic for variables $\vec{x}_A$.

Function *CompNaivePicardOp (A,* $\vec{\mathcal{T}}^0$*, k,* InflLookup*)*

foreach $x \in \vec{x}_A$ **do**

  $\mathcal{T}_x^* \leftarrow f_x$;

  foreach $y \in \vec{y}_x$ **do**

    if $y \in \vec{x}_A$ **then** `/* y is in component A */`

      $\mathcal{T}_x^* \leftarrow \mathcal{T}_x^* \{ y \mapsto \mathsf{InflLookup}(y) \}$;

    else `/* y is solved beforehand, need to extend */`

      $\mathcal{T}_x^* \leftarrow \mathcal{T}_x^* \{ y \mapsto \mathsf{InflLookup}(y) |^{\vec{a}_A} \}$;

    end

  end

  `// substitute the time variable`

  $\mathcal{T}_x^* \leftarrow \mathcal{T}_x^* \{ t \mapsto \tau \}$;

  `// integrate and add initial condition`

  $\mathcal{T}_x^* \leftarrow \mathcal{T}_x^0 |_{\vec{a}_A}|^{t,\vec{a}_A} + \int_0^t \mathcal{T}_x^* d\tau$;

  $\mathcal{T}_x^* \leftarrow \mathtt{LimitOrder}(\mathcal{T}_x^*, k)$;

end

return $\vec{\mathcal{T}}^*$

Algorithm 6: Compositional Picard iteration in naive method

input : reference to the component A, initial conditions $\vec{\mathcal{T}}^0$ for all variables, order k.

precondition : influencers of A contain k-order Picard iterations from 0 to k in the field iter.

postcondition: component A contain k-order Picard iterations from 0 to k in the field iter.

1 **Function** *CompNaivePicardIter (A, $\vec{\mathcal{T}}^0$, k)*
2 **foreach** $x \in \vec{x}_A$ **do**
3 $A.\mathsf{iter}[0]_x \leftarrow \mathcal{T}_x^0|_{\vec{a}_A}|^{(t,\vec{a}_A)}$;
4 **end**
5 **for** $i \leftarrow 1$ **to** k **do**
`// mapping from y to its previous Picard iteration`
6 $\mathsf{InflLookup} \leftarrow \lambda y.\, \mathsf{CompLookup}[y].\mathsf{iter}[i-1]_y$;
7 $A.\mathsf{iter}[i] \leftarrow \mathtt{CompNaivePicardOp}(A, \vec{\mathcal{T}}^0, k, \mathsf{InflLookup})$;
8 **end**

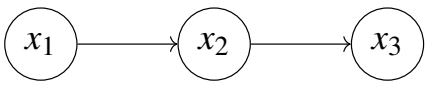


Figure 4.4: Dependency graph.

and Taylor model initial conditions $x(0) = g_{x_1}(a_1)$, $x_2(0) = g_{x_2}(a_2)$, $x_3(0) = g_{x_3}(a_3)$.

The dependency graph for this system is shown in Figure 4.4. We can see that all of the variables are in separate components. Lets call the component where variable x_i is as A_i. The component dependency graph for this system is shown in Figure 4.5, from which we can see that we have to first solve the component A_1, then A_2 and finally A_3.

Let us start with A_1. This component contains the variable x_1 and is not influenced by any variable. The set of parameters relevant to this component is $\vec{a}_{A_1} = \{a_1\}$.

We initialize the Picard iteration for component A_1 as

$$x_1^{(0)}(\vec{a}_{A_1}) = g_{x_1}(a_1).$$

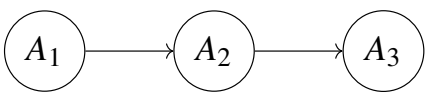


Figure 4.5: Component dependency graph.

Algorithm 7: Compositional integration in naive method

input : reference to the component A, initial conditions $\vec{\mathcal{T}}^0$ for all variables, order k.

precondition : influencers of A contain k-order flow in the field flow.

postcondition: component A contain k-order flow in the field flow.

1 **Function** *CompNaiveIntegrate (A, $\vec{\mathcal{T}}^0$, k)*

`// computing the k-order Picard iteration phase`

2 `CompNaivePicardIter` (A, $(\vec{\mathcal{T}}^0.\mathsf{poly}, [0,0])$, k) ;

`// validating existence and uniqueness of the solution phase`

`// mapping from variable y to its flow`

3 $\mathsf{FlowLookup} \leftarrow \lambda y.\, \mathsf{CompLookup}[y].\mathsf{flow}_y$;

`// guess a remainder interval for the variables` $\vec{x}_A$

4 $\mathbf{i} \leftarrow$ initialGuess ;

5 $A.\mathsf{flow} \leftarrow (A.\mathsf{iter}\ [k]\ , \mathbf{i})$;

6 **while** $A.\mathsf{flow} \not\subseteq$ `CompNaivePicardOp` *(A, $\vec{\mathcal{T}}^0$, k,* FlowLookup*)* **do**

7 $\quad \mathbf{i} \leftarrow \mathrm{enlarge}(\mathbf{i})$;

8 $\quad A.\mathsf{flow} \leftarrow (A.\mathsf{iter}[k], \mathbf{i})$;

9 **end**

`// refining the flow phase`

10 $\vec{\mathcal{T}} \leftarrow A.\mathsf{flow}$;

11 $A.\mathsf{flow} \leftarrow$ `CompNaivePicardOp` (A, $\vec{\mathcal{T}}^0$, k, FlowLookup);

12 **while** $\frac{|W(A.\mathsf{flow.rem})|}{|W(\vec{\mathcal{T}}.\mathsf{rem})|} < threshold$ **do**

13 $\quad \vec{\mathcal{T}} \leftarrow A.\mathsf{flow}$;

14 $\quad A.\mathsf{flow} \leftarrow$ `CompNaivePicardOp` (A, $\vec{\mathcal{T}}^0$, k, FlowLookup);

15 **end**

Algorithm 8: Compositional integration in naive method

input : sorted list of components $\vec{A}$, initial conditions $\vec{\mathcal{T}}^0$ for all variables, order k, time step size Δ.

output: list of k-order flowpipes for each time step

Function *CompNaiveSolve($\vec{A}$, $\vec{\mathcal{T}}^0$, k, Δ)*
  flowpipes $\leftarrow [\,]$;
  for $i \leftarrow 0$ **to** $\lceil \frac{t_{final}}{\Delta} \rceil$ **do**
    $\vec{\mathcal{T}}^{step} \leftarrow 0$;
    foreach $A \in \vec{A}$ **do**
      `CompNaiveIntegrate`$(A, \vec{\mathcal{T}}^0, k)$;
      foreach $x \in \vec{x}_A$ **do**
        $\mathcal{T}_x^{step} \leftarrow A.\text{flow}_x|^{\vec{a}}$;
      end
    end
    flowpipes $\leftarrow$ flowpipes $+ [\vec{\mathcal{T}}]$;
    `// substitute parameter t with its value at the end of the time step, to get initial condition for the next time step`
    $\vec{\mathcal{T}}^0 \leftarrow \vec{\mathcal{T}}^{step}|_{t=\Delta}$;
  end
  return flowpipes;

Continuing on we can compute the next Picard iteration as

$$x_1^{(1)}(t,\vec{a}_{A_1}) = g_{x_1}(a_1) + \int_0^t f_{x_1}(x_1^{(0)}(\vec{a}_{A_1}))d\tau.$$

We can observe that from the first iteration onwards the Picard iteration for variable x only contains the parameters t and $\vec{a}_{A_1}$.

After we have solved A_1, we can solve A_2. This component contains the variable x_2 and is influenced by variable x_1 (from component A_1). The set of parameters relevant to this component is $\vec{a}_{A_2} = \{a_1, a_2\}$.

We initialize the Picard iteration for component A_2 as

$$x_2^{(0)}(a_2) = g_{x_2}(a_2).$$

If we have the Picard iteration for the variable x available, then we can continue to compute the next Picard iteration for y as

$$x_2^{(1)}(t,\vec{a}_{A_2}) = g_{x_2}(a_2) + \int_0^t f_{x_2}(x_1^{(0)}(a_1), x_2^{(0)}(a_2))d\tau.$$

We can again observe that from first iteration onwards the Picard iteration for variable x_2 contains the parameters t and $\vec{a}_{A_2}$.

After we have solved A_2, we can solve A_3. This component contains the variable x_3 and is influenced by variable x_2 (from component A_2). The set of parameters relevant to this component is $\vec{a}_{A_3} = \{a_1, a_2, a_3\}$.

We initialize the Picard iteration for component A_3 as

$$x_3^{(0)}(a_3) = g_{x_3}(a_3).$$

If we have the Picard iteration for the variable x_2 available, then we can continue to compute the next Picard iteration for x_3 as

$$x_3^{(1)}(t,a_2,a_3) = g_{x_3}(a_3) + \int_0^t f_{x_3}(x_2^{(0)}(a_2), x_3^{(0)}(a_3))d\tau.$$

Since x_3 has an influencer with the distance of 2 in the dependency graph we need to compute one more iteration to also include the parameters present in that influencer.

$$x_3^{(2)}(t,\vec{a}_{A_3}) = g_{x_3}(a_3) + \int_0^t f_{x_3}(x_2^{(1)}(\tau,\vec{a}_{A_2}), x_3^{(1)}(\tau,a_2,a_3))d\tau.$$

We can now observe that from the second iteration onwards the Picard iteration for variable x_3 contains the parameters t and $\vec{a}_{A_3}$.

To sum up we make the following observations:

- we can compute the flow for x_1 using a system with 1 variable and 1 parameter,
- we can compute the flow for x_2 using a system with 2 variables and 2 parameters where we need to compute flow for only 1 variable,
- we can compute the flow for x_3 using a system with 2 variables and 3 parameters where we need to compute flow for only 1 variable.

Chapter 5

Compositional Processing of Taylor models

In this chapter, we discuss the processing of Taylor models in the compositional case. As in the general case, Taylor models are modified between the integration phases so that they introduce less over-approximation when they are integrated. We discuss two processing methods: preconditioning and shrink wrapping.

In preconditioning, we discuss how some preconditioned methods are compatible with the compositional approach and how some are not. We give a classification of these methods and argue in detail how only a set of parameters appear in the factored Taylor model with the compatible ones. We also discuss the 2 degrees of compositionality that is possible with them.

In shrink wrapping, we discuss how it is similar to naive method w.r.t. the parameters appearing in the Taylor models, why shrink wrapping itself cannot be made fully compositional and why that is not an important restriction since the integration phase is still fully compositional.

5.1 Compositional Preconditioning

In essence, a preconditioned method introduces less over-approximation by substituting variables in the initial conditions to the EIVP. It can be seen as using a composition of Taylor models $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ to represent the initial conditions where the left model $\vec{\mathcal{U}}$ is the initial condition after variable substitution and the right model $\vec{\mathcal{V}}$ stores the coordinate transformation.

We direct attention to the fact that the left model uses parameters $\vec{b}$ and the right

model uses parameters $\vec{a}$. The parameters $\vec{a}$ are the parameters that appear in the initial conditions for the first time step.

Our goal in this section is to show how some preconditioning methods guarantee how only certain subsets of parameters can appear in the left and right models. In addition to that, we will also argue, like we did in the naive method, that only a subset of parameters can appear in the enclosure computed during the integration phase.

The above mentioned restricted set of parameters for the element x of the left model $\mathcal{U}_x$ is $\vec{b}_x$ and the set of parameters for the element x of the right model $\mathcal{V}_x$ is $\vec{a}_x$. We clarify that the set of parameters $\vec{a}_x$ is defined the same way as we did in the naive method (see Definition 4.4.1).

Definition 5.1.1 (All left model parameters for a variable)**.** We define *all left model parameters of a variable* as

$$\vec{b}_x = \{pv(z) \mid z \in \vec{z}_x\}$$

To be more precise, for the composition of models $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ we show now that $\mathcal{U}_x$, the element of $\vec{\mathcal{U}}$ for variable x, is dependent only on parameters $\vec{b}_x$ and $\mathcal{V}_x$, the element of $\vec{\mathcal{V}}$ for variable x, is dependent only on parameters $\vec{a}_x$. We will show that for a particular class of preconditioning methods these properties hold for all stages of preconditioning method (creating the initial factored Taylor model, after applying the preconditioning and after integration).

Observation 5.1.1. If $z \to^* x$ then $\vec{b}_z \subseteq \vec{b}_x$.

5.1.1 Parameter Dependency in the Factoring of Initial Condition

We remind the reader that typically the EIVP will not have initial conditions represented as a factored Taylor model and we get the first factored Taylor model by applying Algorithm 4.

Let us demonstrate that this will result in the left and the right model satisfying our restriction of parameters.

Theorem 5.1.1 (Parameter dependency in the initial left model)**.** Algorithm 4 produces models where the element corresponding to variable x in the left model $\mathcal{U}_x$ only depends on parameters $\vec{b}_x$.

Proof. By construction, the only parameter appearing in $\mathcal{U}_x$ is $pv(x)$, since $x \in \vec{z}_x$ it also follows that $pv(x) \in \vec{b}_x$. □

Theorem 5.1.2 (Parameter dependency in the initial right model)**.** Algorithm 4 produces models where the element corresponding to a variable x in the right model $\mathcal{V}_x$ only depends on parameters $\vec{a}_x$.

Proof. $\mathcal{V}_x$ is the initial condition for x converted to Taylor model without the constant part. The initial condition for x contains parameters $ip(x)$, since $ip(x) \subseteq \vec{a}_x$ we conclude that the theorem holds. □

5.1.2 Parameter Dependency in the Preconditioning Phase

Let us now investigate what happens with parameters during preconditioning itself.

Suppose that the factorisation that we want to precondition is $\vec{\mathcal{U}}^* \circ \vec{\mathcal{V}}$. In that case, the preconditioning transformation can be expressed by the equation

$$\vec{\mathcal{U}}^* \circ \vec{\mathcal{V}} = \overbrace{(\lambda\vec{b}.\, \vec{c} + \vec{C}\vec{b} + [0,0]^n)}^{\vec{\mathcal{U}}'} \circ \overbrace{(\vec{C}^{-1}(\lambda\vec{b}.\, \vec{T}^*(\vec{b})) \circ \vec{\mathcal{V}})}^{\vec{\mathcal{V}}'}$$

where $\vec{c}$ is the constant part of $\vec{\mathcal{U}}^*$, $\vec{T}^*$ is the non-constant part of $\vec{\mathcal{U}}^*$, $\vec{\mathcal{V}}$ is the right model and $\vec{\mathcal{U}}'$ and $\vec{\mathcal{V}}'$ are the left and the right model after preconditioning.

To make presentation clearer for the compositional setting, we are going differ from (3.8) in two aspects:

- we are using parameters $\vec{b}$ in the left model[1] and parameters $\vec{a}$ in the right model,
- we are grouping the linear term $\vec{C}^*\vec{a}$ and the non-linear term $\vec{n}^*(\vec{a})$ as $\vec{T}^*$.

5.1.2.1 Left model

First, let us inspect the left model $\vec{\mathcal{U}}' = \lambda\vec{b}.\, \vec{c} + \vec{C}\vec{b} + [0,0]^n$ in more detail. We can note that the Taylor model $\vec{\mathcal{U}}'$ will include only the parameters present in the term $\vec{C}\vec{b}$. We can also note that the structure of the matrix $\vec{C}$ determines which parameters are present in the left model after preconditioning. To simplify the presentation, we associate $\vec{C}$'s rows with variables and columns with parameters in the following way

$$\begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \dots & b_n \end{array} \\ \begin{bmatrix} C_{x_1,b_1} & C_{x_1,b_2} & \cdots & C_{x_1,b_n} \\ C_{x_2,b_1} & C_{x_2,b_2} & \cdots & C_{x_2,b_n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{x_n,b_1} & C_{x_n,b_2} & \cdots & C_{x_n,b_n} \end{bmatrix} \end{array}$$

[1] We also use $\vec{b}$ in the right model when moving terms from the left model to the right model.

With this association, we are able to talk about whether parameter b is present in $\mathcal{U}'_x$ by looking at whether the coefficient in the row corresponding to variable x and column corresponding to b in $\vec{C}$ is non-zero or not.

The preconditioning methods of interest to us are the ones which guarantee that only parameters $\vec{b}_x$ appear in $\mathcal{U}'_x$.

Definition 5.1.2 (Compositional preconditioning)**.** We call the preconditioning method *compositional* if the transformation is such that the coefficient $C_{x,b}$ in the matrix for the desired linear part is non-zero only if $b \in \vec{b}_x$.

We also classify preconditioning methods that are not compositional.

Definition 5.1.3 (Non-compositional preconditioning)**.** If a preconditioning method is not compositional then we call it a *non-compositional* preconditioning method.

What is important to us at this point is that applying compositional preconditioning methods restrict the parameters appearing in the left model.

Theorem 5.1.3 (Parameter invariance in the left model in the compositional preconditioning)**.** Let $\vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'$ be the result of compositional preconditioning, then for any variable x the element corresponding to it in the left model $\mathcal{U}'_x$ will only depend on parameters $\vec{b}_x$.

Proof. Left Taylor model after preconditioning is

$$\vec{\mathcal{U}}' = (\lambda\vec{b}.\, \vec{c} + \vec{C}\vec{b} + [0,0]^n)$$

where the element corresponding to x is

$$\mathcal{U}'_x = (\lambda\vec{b}.\, c_x + C_x\vec{b} + [0,0])$$

where C_x is the row corresponding to x in $\vec{C}$. The term $C_x\vec{b}$ can be expressed as

$$C_x\vec{b} = \sum_{i=1}^{n} C_{x_i,b_i} b_i$$

which we can rewrite as

$$C_x\vec{b} = \sum_{b_i \in \vec{b}_x} C_{x_i,b_i} b_i + \sum_{b_i \notin \vec{b}_x} C_{x_i,b_i} b_i$$

where we know that all of the coefficients of the right sum are zero. That is, only parameters $\vec{b}_x$ may appear in $\mathcal{U}'_x$. □

5.1.2.2 Right model

The right model after preconditioning $\vec{\mathcal{V}}'$ depends on the left model (in addition to the right model) before preconditioning $\vec{\mathcal{U}}$. With that in mind, let us assume that the left model is such that $\mathcal{U}_x$ only depends on parameters $\vec{b}_x$ from this point forward. Our aim in this section is to show how this results that only some parameters will appear in the right model after preconditioning.

We remind the reader that the right model after preconditioning has the form $\vec{\mathcal{V}}' = \vec{C}^{-1}(\lambda\vec{b}.\ \vec{\mathcal{T}}^*(\vec{b})) \circ \vec{\mathcal{V}}$. To see which parameters are relevant for a specific element of it, we are going to apply or the operations here.

To start with, we associate $\vec{C}^{-1}$'s rows with variables and columns with parameters in a similar fashion as we did with $\vec{C}$

$$\begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \dots & b_n \end{array} \\ \begin{bmatrix} k_{x_1,b_1} & k_{x_1,b_2} & \cdots & k_{x_1,b_n} \\ k_{x_2,b_1} & k_{x_2,b_2} & \cdots & k_{x_2,b_n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{x_n,b_1} & k_{x_n,b_2} & \cdots & k_{x_n,b_n} \end{bmatrix} \end{array} \tag{5.1}$$

The preconditioning methods interest to us now are the ones which guarantee that only parameters $\vec{a}_x$ appear in $\mathcal{V}'_x$.

We will now prove[2] that if we are dealing with compositional preconditioning the coefficient $k_{x,b}$ in the inversion of the matrix $\vec{C}$ for the desired linear part is non-zero only if $b \in \vec{b}_x$.

We can do that by viewing $\to^*$ as a binary relation on $\{1,\dots,n\}$.

An $n \times n$ matrix A *respects* $\to^*$ if whenever $a_{ij} \neq 0$, then $j \to^* i$. That is each row i of A only has non-zero entries at columns j for which $j \to^* i$. That is, we say that $\vec{C}$ respects $\to^*$ and if $j \to^* i$ then we have that the variable x_j is an influencer of the variable x_i. To have restrictions on the right model after preconditioning, we need to have restrictions on $\vec{C}^{-1}$. That is, we need to show that if a matrix A respects $\to^*$, then so does its inversion.

Lemma 5.1.4. Identity matrix respects $\to^*$.

Proof. Immediate □

[2] This proof is from personal discussion with Paul B. Jackson

Lemma 5.1.5. If $n \times n$ matrix A respects $\rightarrow^*$, then kA also respects $\rightarrow^*$ for any scalar value k.

Proof. Immediate □

Lemma 5.1.6. If $n \times n$ matrices A, B respects $\rightarrow^*$, then $A + B$ also respects $\rightarrow^*$.

Proof. Immediate □

Lemma 5.1.7. If $n \times n$ matrices A, B respects $\rightarrow^*$, then AB also respects $\rightarrow^*$.

Proof. Let A, B be such that they respect $\rightarrow^*$ and let $C = [c_{ij}] = AB$. Let us fix i and j and let us assume that $c_{ij} \neq 0$.

We have to show that $j \rightarrow^* i$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

For $c_{ij} \neq 0$, we must have at least one of the terms $a_{ik} b_{kj} \neq 0$. Consider one such term. For this term, it must be that both $a_{ik} \neq 0$ and $b_{kj} \neq 0$. Since both A and B respect $\rightarrow^*$, we must then have $k \rightarrow^* i$ and $j \rightarrow^* k$. Since $\rightarrow^*$ is transitive, we also have $j \rightarrow^* i$ which is what we needed to show. □

Theorem 5.1.8 (Cayley-Hamilton theorem)**.** If a $n \times n$ matrix A is invertible, then there exist scalars $k_0, \ldots, k_{n-1}$ such that

$$A^{-1} = \sum_{i=0}^{n-1} k_i A^i$$

Theorem 5.1.9. If $n \times n$ matrices A respects $\rightarrow^*$ and is invertible, then A^{-1} also respects $\rightarrow^*$.

Proof. From Theorem 5.1.8 we know that we can express A^{-1} as

$$A^{-1} = \sum_{i=0}^{n-1} k_i A^i.$$

Using Lemmas 5.1.4, 5.1.5, 5.1.6 and 5.1.7 we can see that all of the operations involved in computing A^{-1} respect $\rightarrow^*$. □

Theorem 5.1.9 together with the discussion above gives us an observation about non-zero elements in the matrix C and C^{-1}.

Observation 5.1.2. In compositional preconditioning methods both the coefficient $C_{x,b}$ in the matrix for the desired linear part and the coefficient $k_{x,b}$ in its inversion are non-zero only if $b \in \vec{b}_x$.

We can make use of this form on C^{-1} to guarantee that some of the parameters will not be present for the Taylor models for some variables.

Theorem 5.1.10 (Parameter invariance in the right model in the compositional preconditioning)**.** Let $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ be the input and $\vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'$ be the output of compositional preconditioning. Then if $\mathcal{U}_x$ only depends on parameters $\vec{b}_x$ and $\mathcal{V}_x$ only depends on parameters $\vec{a}_x$ for any x, it follows that $\mathcal{V}'_x$ only depends on parameters $\vec{a}_x$.

Proof. The right Taylor model after preconditioning is

$$\vec{\mathcal{V}}' = \vec{C}^{-1}(\lambda\vec{b}.\ \vec{\mathcal{T}}^*(\vec{b})) \circ \vec{\mathcal{V}}.$$

Let us focus on only the element corresponding to x in $\vec{\mathcal{V}}'$

$$\mathcal{V}'_x = C_x^{-1}(\lambda\vec{b}.\ \vec{\mathcal{T}}^*(\vec{b})) \circ \vec{\mathcal{V}}.$$

Considering the form of C^{-1} (5.1) we can expand it as

$$\mathcal{V}'_x = (\lambda\vec{b}.\ k_{x,b_1}\mathcal{T}^*_{x_1}(\vec{b}) + \cdots + k_{x,b_n}\mathcal{T}^*_{x_n}(\vec{b})) \circ \vec{\mathcal{V}}.$$

From the assumption of the theorem, we know that each $\mathcal{T}^*_{x_i}$ only depends on parameters $\vec{b}_{x_i}$. We make this explicit by introducing a version $\mathcal{T}^*_{x_i}$ of the operator $\mathcal{T}^*_{x_i}$ that takes as argument $\vec{b}_{x_i}$ rather than $\vec{b}$, and write

$$\mathcal{V}'_x = (\lambda\vec{b}.\ k_{x,b_1}\mathcal{T}^*_{x_1}(\vec{b}_{x_1}) + \cdots + k_{x,b_n}\mathcal{T}^*_{x_n}(\vec{b}_{x_n})) \circ \vec{\mathcal{V}}.$$

We can note that since the preconditioning method is assumed to be compositional that the coefficients k_{x,b_i}s are non-zero only if x_i is an influencer of x. Let $\vec{z}_x$, the set of influencers for x be $\{z_1, \ldots, z_p\}$. Considering that $x_i \notin \vec{z}_x \Rightarrow k_{x,pv(x_i)} = 0$ this means that we get

$$\mathcal{V}'_x = (\lambda\vec{b}.\ k_{x,pv(z_1)}\mathcal{T}^*_{z_1}(\vec{b}_{z_1}) + \cdots + k_{x,pv(z_p)}\mathcal{T}^*_{z_p}(\vec{b}_{z_p})) \circ \vec{\mathcal{V}}.$$

Keeping in mind that $\vec{b}_{z_i} \subseteq \vec{b}_x$, we can create extended versions $\mathcal{T}^*_{z_i}|^{\vec{b}_x}$ of the operators $\mathcal{T}^*_{z_i}$ that take arguments $\vec{b}_x$ rather than $\vec{b}_{z_i}$. We then have

$$\mathcal{V}'_x = (\lambda\vec{b}_x.\ k_{x,b_{z_1}}\mathcal{T}^*_{z_1}|^{\vec{b}_x}(\vec{b}_x) + \cdots + k_{x,b_{z_p}}\mathcal{T}^*_{z_p}|^{\vec{b}_x}(\vec{b}_x)) \circ \vec{\mathcal{V}}_{\vec{z}_x}|^{\vec{a}_x} \tag{5.2}$$

where $\vec{\mathcal{V}}_{\vec{z}_x}$ is the vector of elements $\mathcal{V}_z$ of $\vec{\mathcal{V}}$ such that $z \to^* x$ and $\mathcal{V}_{\vec{z}_x}|^{\vec{a}_x}$ has each of these $\mathcal{V}_z$ replaced by a extended version $\mathcal{V}_z|^{\vec{a}_x}$. We note that the occurrences of $\vec{a}_x$ here should not be considered free, capable of being bound by some λ.

We are now ready to show how a particular compositional structure of the right Taylor model is preserved by preconditioning. By our assumption $\mathcal{V}_x$ has parameters $\vec{a}_x$ and if $z \to^* x$ then $\vec{a}_z \subseteq \vec{a}_x$. From equation (5.2) above, the $\vec{a}$ parameters involved in $\mathcal{V}'_x$ are

$$\bigcup_{z \to^* x} \vec{a}_z.$$

But since we have that all such $\vec{a}_z \subseteq \vec{a}_x$, the parameters involved in $\mathcal{V}'_x$ are just $\vec{a}_x$.

□

5.1.3 Parameter Dependency in the Integration of the Left Model

In preconditioned methods the preconditioning is interleaved with integration, so far we have shown that if we are using a compositional preconditioning method, then we retain the absence of some parameters in both the left model and the right model after preconditioning. To guarantee that this property holds for the whole cycle, we need to show that it also holds for the integration phase.

We also note that if we wish to integrate a factored Taylor model $\vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'$ then we only need to integrate the left model $\vec{\mathcal{U}}'$ and compose the result $\vec{\mathcal{U}}^*$ with the right model $\vec{\mathcal{V}}'$.

Theorem 5.1.11 (Parameter invariance in the flow of the compositional preconditioning)**.** Suppose that the input of integration is $\vec{\mathcal{U}}' \circ \vec{\mathcal{V}}'$ where $\vec{\mathcal{U}}'$ is such that that for any variable x the element $\mathcal{U}'_x$ only depends on parameters $\vec{b}_x$. Then the integration result $\vec{\mathcal{U}}^* \circ$ is such that for any variable x the element $\mathcal{U}^*_x$ will only depend on parameters $(t, \vec{b}_x)$.

Proof. The right model is unaffected by integration and can be ignored.

The elements of the left model $\vec{\mathcal{U}}'$ serve as initial conditions for their corresponding variables i.e. $\mathcal{U}'_x$ is the initial condition for variable x. Let us introduce symbolic initial condition parameters to represent each initial conditions i.e. the initial condition parameter for variable x is x^0.

Let us blur the distinction between integration and Picard iteration[3], which enables us to use Theorem 4.3.3. That theorem states that the element corresponding to vari-

[3] With preconditioned integration, the argument is the same as was in the naive method.

able x in the Taylor model computed in the integration can be expressed with only parameters t and initial condition parameters $\vec{z}_x^0$ i.e. $\mathcal{U}_x^*(t,\vec{z}_x^0)$.

We can express $\mathcal{U}_x^*$ using parameters $\vec{b}$ instead of $\vec{x}^0$ if we replace each initial condition parameter with the Taylor model expressing its value in terms of the parameters $\vec{b}$. That is, we substitute x^0 with $\mathcal{U}_x'$ in $\mathcal{U}_x^*$.

Consider some parameter z^0 present in $\vec{z}_x^0$. If we substitute z^0 with $\mathcal{U}_z'$ then we introduce parameters present in $\mathcal{U}_z'$. From Theorem 5.1.3 we know that these parameters are $\vec{b}_z$.

Furthermore, from the definition of $\vec{z}_x^0$ we know that if $z^0 \in \vec{z}_x^0$ then this means that z is an influencer of x i.e. $z \to^* x$ and we can conclude using Observation 5.1.1 that $\vec{b}_z \subseteq \vec{b}_x$.

So the replacement of any parameter z^0 results in the introduction of parameters within $\vec{b}_x$, and we, therefore, have that $\mathcal{U}_x^*$ only depends on parameters $\vec{b}_x$ (in addition to the time parameter t). □

We note that if the integration result $\vec{\mathcal{U}}'$ only depends on parameters $(t,\vec{b}_x)$, then after we replace t with the value t_{final}, we get a model which only contains parameters $\vec{b}_x$.

5.1.4 Parameter Dependency Examples in Preconditioning Methods

We give examples describing the parameter dependencies in non-compositional preconditioning and compositional preconditioning.

Example 16 (Parameter dependencies in non-compositional preconditioning)**.** We illustrate the parameter dependency in non-compositional preconditioning with the EIVP with the ODEs

$$\begin{aligned}\frac{dx_1}{dt} &= f_{x_1}(x_1)\\ \frac{dx_2}{dt} &= f_{x_2}(x_1,x_2)\\ \frac{dx_3}{dt} &= f_{x_3}(x_2,x_3)\end{aligned}$$

and initial conditions the same as what was used in Example 12. From where we know that we can represent initial conditions as

$$\begin{bmatrix}x_1(0)\\ x_2(0)\\ x_3(0)\end{bmatrix} = \overbrace{\begin{bmatrix}\frac{1}{2}+b_1+[0,0]\\ \frac{3}{2}+b_2+[0,0]\\ 1+b_3+[0,0]\end{bmatrix}}^{\vec{\mathcal{U}}} \circ \overbrace{\begin{bmatrix}\frac{1}{2}a_1+[0,0]\\ \frac{1}{2}a_2+[0,0]\\ a_3+[0,0]\end{bmatrix}}^{\vec{\mathcal{V}}} \circ D.$$

Since we are only interested in the parameter dependency we abstract away from scalars and represent the left and right Taylor models as

$$\vec{\mathcal{U}} \circ \vec{\mathcal{V}} = \left(\begin{bmatrix} c_{x_1} \\ c_{x_2} \\ c_{x_3} \end{bmatrix} + \begin{bmatrix} \mathcal{T}_{x_1}(b_1) \\ \mathcal{T}_{x_2}(b_2) \\ \mathcal{T}_{x_3}(b_3) \end{bmatrix} \right) \circ \begin{bmatrix} \mathcal{V}_{x_1}(a_1) \\ \mathcal{V}_{x_2}(a_2) \\ \mathcal{V}_{x_3}(a_3) \end{bmatrix}.$$

Let us also assume that our desired linear part of the new left Taylor model $\vec{C}$ and its inverse $\vec{C}^{-1}$ are

$$\vec{C} = \begin{bmatrix} C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,1} & C_{3,2} & C_{3,3} \end{bmatrix}, \quad \vec{C}^{-1} = \begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}.$$

Using these $\vec{C}$ and $\vec{C}^{-1}$, we get the new left Taylor model

$$\vec{\mathcal{U}}' = \begin{bmatrix} c_{x_1} \\ c_{x_2} \\ c_{x_3} \end{bmatrix} + \begin{bmatrix} C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,1} & C_{3,2} & C_{3,3} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

which we can rewrite as

$$\vec{\mathcal{U}}' = \begin{bmatrix} c_{x_1} + C_{1,1}b_1 + C_{1,2}b_2 + C_{1,3}b_3 \\ c_{x_2} + C_{2,1}b_1 + C_{2,2}b_2 + C_{2,3}b_3 \\ c_{x_2} + C_{3,1}b_1 + C_{3,2}b_2 + C_{3,3}b_3 \end{bmatrix}. \tag{5.3}$$

Therefore in the general case[4], we get that

$$\vec{\mathcal{U}}' = \begin{bmatrix} \mathcal{U}'_{x_1}(\vec{b}) \\ \mathcal{U}'_{x_2}(\vec{b}) \\ \mathcal{U}'_{x_3}(\vec{b}) \end{bmatrix}$$

i.e. any element of $\vec{\mathcal{U}}'$ can contain any parameter.

Similarly we get the new right Taylor model as

$$\vec{\mathcal{V}}' = \left(\begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} \begin{bmatrix} \mathcal{T}_{x_1}(b_1) \\ \mathcal{T}_{x_2}(b_2) \\ \mathcal{T}_{x_3}(b_3) \end{bmatrix} \right) \circ \begin{bmatrix} \mathcal{V}_{x_1}(a_1) \\ \mathcal{V}_{x_2}(a_2) \\ \mathcal{V}_{x_3}(a_3) \end{bmatrix}$$

which can be rewritten as

$$\vec{\mathcal{V}}' = \begin{bmatrix} k_{1,1}\mathcal{T}_{x_1}(b_1) + k_{1,2}\mathcal{T}_{x_2}(b_2) + k_{1,3}\mathcal{T}_{x_3}(b_3) \\ k_{2,1}\mathcal{T}_{x_1}(b_1) + k_{2,2}\mathcal{T}_{x_2}(b_2) + k_{2,3}\mathcal{T}_{x_3}(b_3) \\ k_{3,1}\mathcal{T}_{x_1}(b_1) + k_{3,2}\mathcal{T}_{x_2}(b_2) + k_{3,3}\mathcal{T}_{x_3}(b_3) \end{bmatrix} \circ \begin{bmatrix} \mathcal{V}_{x_1}(a_1) \\ \mathcal{V}_{x_2}(a_2) \\ \mathcal{V}_{x_3}(a_3) \end{bmatrix}. \tag{5.4}$$

[4]The case where we do not have any restriction on the matrix C.

If we denote the left part of this composition as $\vec{\mathcal{T}}'$, we get

$$\vec{\mathcal{V}}' = \begin{bmatrix} \mathcal{T}'_{x_1}(b_1,b_2,b_3) \\ \mathcal{T}'_{x_2}(b_1,b_2,b_3) \\ \mathcal{T}'_{x_3}(b_1,b_2,b_3) \end{bmatrix} \circ \begin{bmatrix} \mathcal{V}_{x_1}(a_1) \\ \mathcal{V}_{x_2}(a_2) \\ \mathcal{V}_{x_3}(a_3) \end{bmatrix}$$

where we can apply the composition and get

$$\vec{\mathcal{V}}' = \begin{bmatrix} \mathcal{T}'_{x_1}(\mathcal{V}_{x_1}(a_1),\mathcal{V}_{x_2}(a_2),\mathcal{V}_{x_3}(a_3)) \\ \mathcal{T}'_{x_2}(\mathcal{V}_{x_1}(a_1),\mathcal{V}_{x_2}(a_2),\mathcal{V}_{x_3}(a_3)) \\ \mathcal{T}'_{x_3}(\mathcal{V}_{x_1}(a_1),\mathcal{V}_{x_2}(a_2),\mathcal{V}_{x_3}(a_3)) \end{bmatrix} = \begin{bmatrix} \mathcal{V}'_{x_1}(\vec{a}) \\ \mathcal{V}'_{x_2}(\vec{a}) \\ \mathcal{V}'_{x_3}(\vec{a}) \end{bmatrix}.$$

In conclusion, we can say that we cannot eliminate any parameters from neither left nor right model if we apply non-compositional preconditioning to this example system.

Example 17 (Parameter dependencies in compositional preconditioning)**.** Let us assume that we are using the same system as we did in Example 16. We can reason about compositional methods exactly as we did in Example 16 up to the equation (5.3) which we can simplify since we know that in the compositional methods the coefficients $C_{1,2}$, $C_{1,3}$ and $C_{2,3}$ are zero. If we take this into account, we may rewrite the left model $\vec{\mathcal{U}}'$ as

$$\vec{\mathcal{U}}' = \begin{bmatrix} c_{x_1} + C_{1,1}b_1 \\ c_{x_2} + C_{2,1}b_1 + C_{2,2}b_2 \\ c_{x_2} + C_{3,1}b_1 + C_{3,2}b_2 + C_{3,3}b_3 \end{bmatrix}.$$

We can see that some of the elements do not include all of the parameters. Namely, the left model becomes

$$\vec{\mathcal{U}}' = \begin{bmatrix} \mathcal{U}'_{x_1}(b_1) \\ \mathcal{U}'_{x_2}(b_1,b_2) \\ \mathcal{U}'_{x_3}(b_1,b_2,b_3) \end{bmatrix}.$$

Now let us look at the right model. Our interest in this example is to demonstrate how only some parameters can be present in the preconditioned models. With that in mind, we are going to allow more parameters in our initial model. This is justified since if we arrive at some restrictions using these models, we can argue that those restrictions will hold if the initial models do not include extra parameters. The relaxation we are going to use is that we are going to use the set $\vec{b}_{x_i}$ instead of b_i in the left model and set $\vec{a}_{x_i}$ instead of a_i in the right model. The motivation for using these more relaxed

parameters is that the restriction we illustrate will hold even if we integrate before the preconditioning.

We present these relaxed sets $\vec{b}_{x_i}$ and $\vec{a}_{x_i}$ for each of the variables in this system in the following table.

i	1	2	3
$\vec{b}_{x_i}$	$\{b_1\}$	$\{b_1, b_2\}$	$\{b_1, b_2, b_3\}$
$\vec{a}_{x_i}$	$\{a_1\}$	$\{a_1, a_2\}$	$\{a_1, a_2, a_3\}$

With the previous discussion in mind, the input that we are going to use in preconditioning is

$$\vec{\mathcal{U}} \circ \vec{\mathcal{V}} = \left(\begin{bmatrix} c_{x_1} \\ c_{x_2} \\ c_{x_3} \end{bmatrix} + \begin{bmatrix} \mathcal{T}_{x_1}(\vec{b}_{x_1}) \\ \mathcal{T}_{x_2}(\vec{b}_{x_2}) \\ \mathcal{T}_{x_3}(\vec{b}_{x_3}) \end{bmatrix} \right) \circ \begin{bmatrix} \mathcal{V}_{x_1}(\vec{a}_{x_1}) \\ \mathcal{V}_{x_2}(\vec{a}_{x_2}) \\ \mathcal{V}_{x_3}(\vec{a}_{x_3}) \end{bmatrix}.$$

If we add extra parameters where appropriate, then we can reason about the right model as we did in Example 16 up to the equation (5.4). We can simplify the equation (5.4) here since we know that in the compositional methods the coefficients $k_{1,2}$, $k_{1,3}$ and $k_{2,3}$ are zero. If we take this into account, we may rewrite the right model as

$$\vec{\mathcal{V}}' = \begin{bmatrix} k_{1,1}\mathcal{T}_{x_1}(\vec{b}_{x_1}) \\ k_{2,1}\mathcal{T}_{x_1}(\vec{b}_{x_1}) + k_{2,2}\mathcal{T}_{x_2}(\vec{b}_{x_2}) \\ k_{3,1}\mathcal{T}_{x_1}(\vec{b}_{x_1}) + k_{3,2}\mathcal{T}_{x_2}(\vec{b}_{x_2}) + k_{3,3}\mathcal{T}_{x_3}(\vec{b}_{x_3}) \end{bmatrix} \circ \begin{bmatrix} \mathcal{V}_{x_1}(\vec{a}_{x_1}) \\ \mathcal{V}_{x_2}(\vec{a}_{x_2}) \\ \mathcal{V}_{x_3}(\vec{a}_{x_3}) \end{bmatrix}.$$

If we denote the left part of this composition as $\vec{\mathcal{T}}'$ and use the property that $\vec{b}_{x_1} \subset \vec{b}_{x_2} \subset \vec{b}_{x_3}$, we get

$$\vec{\mathcal{V}}' = \begin{bmatrix} \mathcal{T}'_{x_1}(\vec{b}_{x_1}) \\ \mathcal{T}'_{x_2}(\vec{b}_{x_2}) \\ \mathcal{T}'_{x_3}(\vec{b}_{x_3}) \end{bmatrix} \circ \begin{bmatrix} \mathcal{V}_{x_1}(\vec{a}_{x_1}) \\ \mathcal{V}_{x_2}(\vec{a}_{x_2}) \\ \mathcal{V}_{x_3}(a_3) \end{bmatrix}$$

and if we apply the composition together with the property that $\vec{a}_{x_1} \subset \vec{a}_{x_2} \subset \vec{a}_{x_3}$, we get

$$\vec{\mathcal{V}}' = \begin{bmatrix} \mathcal{T}'_{x_1}(\mathcal{V}_{x_1}(\vec{a}_{x_1})) \\ \mathcal{T}'_{x_2}(\mathcal{V}_{x_1}(\vec{a}_{x_1}), \mathcal{V}_{x_2}(\vec{a}_{x_2})) \\ \mathcal{T}'_{x_3}(\mathcal{V}_{x_1}(\vec{a}_{x_1}), \mathcal{V}_{x_2}(\vec{a}_{x_2}), \mathcal{V}_{x_3}(\vec{a}_{x_3})) \end{bmatrix} = \begin{bmatrix} \mathcal{V}'_{x_1}(\vec{a}_{x_1}) \\ \mathcal{V}'_{x_2}(\vec{a}_{x_2}) \\ \mathcal{V}'_{x_3}(\vec{a}_{x_3}) \end{bmatrix}.$$

In conclusion, we can say that if we apply compositional preconditioning to the example system, we can restrict the set of parameters appearing in both the left and the right model.

5.1.5 Computing Flowpipes with Compositional Preconditioning

In the previous section, we have shown that compositional preconditioning methods restrict the possible parameters appearing in both the left model and also in the right model. We can make use of this property in two ways:

- using smaller dimensional systems during integration, but using the original system during preconditioning,
- using smaller dimensional systems during integration and preconditioning.

The reason why it makes sense to consider partial composition (or none at all for that matter), is that while composition enables the use of smaller data structures it also introduces some overhead. The overhead introduced lies in transforming flowpipes so that they would include all the needed parameters when using them in later components than where they were computed. Compared to the naive compositional algorithm, the preconditioned algorithm needs more transformation.

We describe the above mentioned two approaches in greater details in this section. During this section, let us suppose that we have an EIVP $(\vec{f}, \vec{X}_{init}, [0, t_{final}])$ where the initial condition is represented by factored Taylor model i.e. $\vec{X}_{init} = \vec{\mathcal{U}} \circ \vec{\mathcal{V}}$. Suppose we also have a partially ordered set of components $(\vec{A}, \preceq)$ and a sorting on components $\vec{A}$. We will describe how we can compute flowpipes for the components.

5.1.5.1 Compositional integration with non-compositional preconditioning

Our first option is to use composition during integration but not during preconditioning. To be able to do that we need to use compositional preconditioning methods during preconditioning, let us assume that it is the case.

Since we are not using components during preconditioning we can ignore preconditioning itself and focus only on integration.

We know from the above discussion that if we use compositional preconditioning methods then $\vec{\mathcal{U}}$, $\vec{\mathcal{U}}'$ and $(\vec{\mathcal{U}}^*|_{t=t_{final}})$ depend on parameters $\vec{b}_x$ (Theorem 5.1.1, Theorem 5.1.3 and Theorem 5.1.11 after substituting t) and the model $\vec{\mathcal{U}}^*$ depends on parameters $(t, \vec{b}_x)$ (Theorem 5.1.11).

Let us now discuss how we could use these restricted sets of parameters.

We can again view integration as computing Picard iterations. Let us remind that in the preconditioned method the initial conditions used in the Picard iteration are the elements of $\vec{\mathcal{U}}'(\vec{b})$.

In the general case, we get that the Picard iteration for variable x is an operator of $(t,\vec{b})$

$$x^{(j)}(t,\vec{b}) = \mathcal{U}'_x(\vec{b}) + \int_0^t f_x(y_1^{(j-1)}(\tau,\vec{b}),\ldots,y_m^{(j-1)}(\tau,\vec{b}))d\tau$$

with $x^{(0)}(t,\vec{b}) = \mathcal{U}'_x(\vec{b})$.

Using Theorem 5.1.11 and defining $\vec{b}_A = \vec{b}_x$ we can argue as we did in the naive method, that we can view this as an operator with arguments $(t,\vec{b}_A)$, i.e. we get

$$x^{(j)}(t,\vec{b}_A) = \mathcal{U}'_x|_{\vec{b}_A}|^{t,\vec{a}_A}(t,\vec{a}_A) + \int_0^t f_x(y_1^{(j-1)}|^{t,\vec{b}_A}(\tau,\vec{b}_A),\ldots,y_m^{(j-1)}|^{t,\vec{b}_A}(\tau,\vec{b}_A)d\tau \quad (5.5)$$

with $x^{(0)}(t,\vec{b}_A) = \mathcal{U}'_x|_{\vec{b}_A}|^{t,\vec{a}_A}(t,\vec{a}_A)$.

Let us note again that on the right side of this equation we need the initial condition of x as well as the $(j-1)^{th}$ Picard iterations of the immediate influencers of x. Both of these are available if we solve components w.r.t. to the order $\preceq$.

We can also use the same reasoning to define a compositional Picard operator for a component in the preconditioned method.

And we can also note again that computing flows for component A can be viewed as solving an EIVP which has variables $\vec{x}_A$ whose initial conditions have parameters $\vec{b}_A$ giving us a lower-dimensional system.

We note that since $\vec{b}_x \subseteq \vec{b}$ we can extend the flow $\mathcal{U}^*_x(t,\vec{b}_x)$ to be an operator with an argument $(t,\vec{b})$. We can use this extended flow as the input to the preconditioning in the next step. That is, if the flow for variable x is $\mathcal{U}^*_x(t,\vec{b}_x)\circ\vec{\mathcal{V}}'$ then the initial step for the integration after t_{final} will be $((\mathcal{U}^*_x(t,\vec{b}_x)|_{t=t_{final}})|^{\vec{b}})\circ\vec{\mathcal{V}}'$.

For the preconditioned method, the pseudo-code for the compositional Picard operator, the pseudo-code the compositional Picard iteration and the pseudo-code for compositional integration are essentially copies of the pseudo-codes of the corresponding methods in the naive method. The only difference is that since the initial conditions used in integration in the naive method depend on the parameters $\vec{a}$ and the initial conditions in preconditioned integration depend on parameters $\vec{b}$, therefore we need to use restriction $|_{\vec{b}_A}$ instead of $|_{\vec{a}_A}$ and extension $|^{t,\vec{b}_A}$ instead of $|^{t,\vec{a}_A}$.

We give the pseudo-code for computing the flow for EIVP problem in Algorithm 9. The algorithm uses compositional integration `CompPrecondIntegrate` (given in the Algorithm 7) which is a copy of `CompNaiveIntegrate` with the exception that it uses the functions `CompPrecondPicardIter` and `CompPrecondPicardOp` instead of the functions `CompNaivePicardIter` and `CompNaivePicardOp`. The functions `CompPrecondPicardIter`

and `CompPrecondPicardOp` are copies of `CompNaivePicardIter` and `CompNaivePicardOp`, respectively, with the exception that restricting and extending is done with respect to the parameters $\vec{b}_x$ instead of the parameters $\vec{a}_x$.

We note that `CompPrecondIntegrate` integrates with the initial condition $\vec{\mathcal{U}}'$ and stores the result in the field `flow` for each component.

For non-compositional preconditioning and compositional integration, we give the pseudo-code for integration over multiple time steps in Algorithm 10.

Algorithm 9: Computing flow with compositional integration and non-compositional preconditioning.

input : sorted list of components $\vec{A}$, initial conditions $(\vec{\mathcal{U}}, \vec{\mathcal{V}})$ for all variables, order k.

output: k-order flow for all the variables in the system.

```
1 Function LeftCompPrecondIntegrate (A⃗, (U⃗, V⃗), k)
      // precondition the initial conditions for the whole system
2     (U⃗', V⃗') ← Precondition(U⃗, V⃗) ;
3     foreach A ∈ A⃗ do
          // integrate each component separately and store the
             flow in the field 'flow'
4         CompPrecondIntegrate (A, U⃗', k) ;
          // extend the domain of the flow back
5         foreach x ∈ x⃗_A do
6             U*_x ← A.flow_x|^{t,a⃗} ;
7         end
8     end
9     return (U⃗*, V⃗')
```

5.1.6 Compositional Integration with Compositional Preconditioning

Our second option is to use composition during both integration and preconditioning. To be able to do that we again need to use compositional preconditioning methods during preconditioning, let us assume again that it is the case.

Algorithm 10: Compositional integration and non-compositional preconditioning over multiple time steps

input : sorted list of components $\vec{A}$, factored initial conditions $(\vec{\mathcal{U}}^0, \vec{\mathcal{V}}^0)$ for all variables, order k, time step size Δ.

output: list of factored k-order flowpipes for each time step

Function *LeftCompPrecondSolve*($\vec{A}$, $(\vec{\mathcal{U}}^0, \vec{\mathcal{V}}^0)$, k, Δ)
  flowpipes $\leftarrow [\,]$;
  for $i \leftarrow 0$ **to** $\lceil \frac{t_{final}}{\Delta} \rceil$ **do**
    $(\vec{\mathcal{U}}^*, \vec{\mathcal{V}}') \leftarrow$ `LeftCompPrecondIntegrate`$(\vec{A}, (\vec{\mathcal{U}}^0, \vec{\mathcal{V}}^0), k)$;
    flowpipes $\leftarrow$ flowpipes $+ [(\vec{\mathcal{U}}^*, \vec{\mathcal{V}}')]$;
    `// substitute parameter` t `in left model`
    $(\vec{\mathcal{U}}^0, \vec{\mathcal{V}}^0) \leftarrow (\vec{\mathcal{U}}^*|_{t=\Delta}, \vec{\mathcal{V}}')$;
  end
  return flowpipes;

From (3.8), the left model for variable x is

$$\mathcal{U}'_x = (\lambda \vec{b}.\ c_x + C_x \vec{b} + [0,0]).$$

We can see that we can compute this Taylor model if we know the constant of $\mathcal{U}_x$ and the row corresponding to x in the matrix C. Let us assume that we have these assumptions satisfied[5].

Let us now look at the right Taylor model. We remind the reader the equation (5.2) for computing the element of the right model corresponding to variable x has the form

$$\mathcal{V}'_x = (\lambda \vec{b}_x.\ k_{x,b_{z_1}} \mathcal{T}^*_{z_1}|^{\vec{b}_x}(\vec{b}_x) + \cdots + k_{x,b_{z_p}} \mathcal{T}^*_{z_p}|^{\vec{b}_x}(\vec{b}_x)) \circ \vec{\mathcal{V}}_{\vec{z}_x}|^{\vec{a}_x}.$$

Let us note that to compute the right model for variable x we need to know:

1. the row of the matrix C^{-1} corresponding to x,
2. the non-constant parts $\mathcal{T}^*_{z_i}$s of the elements of $\vec{\mathcal{U}}$ corresponding to the influencers of x,
3. the elements of $\vec{\mathcal{V}}$ corresponding to the influencers of x.

[5] Identity preconditioning is an example of a compositional method where we know the row C_x

Since the influencers of x are either in the same component as x or in the preceding ones, we can assume that we have them available for both $\vec{\mathcal{U}}$ and $\vec{\mathcal{V}}$. Let us also assume that we the coefficients of the row C^{-1} available[6].

We give the pseudo-code for compositional preconditioning in Algorithm 11 and the pseudo-code for computing the flow with compositional integration and compositional preconditioning in Algorithm 12. In there, the components are augmented with the fields $\vec{\mathcal{U}}, \vec{\mathcal{V}}, \vec{\mathcal{U}}'$ and $\vec{\mathcal{V}}'$ where the left model before preconditioning, the right model before preconditioning, the left model after preconditioning and the right model after preconditioning are stored respectively. The flow computed for a component in Algorithm 12 is a composition of the fields `flow` and $\vec{\mathcal{V}}'$.

In Algorithm 11, the function `DesiredLinearPart` computes the part of the matrix $\vec{C}$ relevant to a component A, the function `NonConstantPart` computes the non-constant part of the Taylor model given as argument, pv is the left model parameter function (Definition 2.2.18) and pv^{-1} its inverse.

For compositional integration and compositional preconditioning, we give the pseudo-code for integration over multiple time steps in Algorithm 13.

5.1.7 Classes of Preconditioning Methods

We conclude this section by presenting the classes of the preconditioning methods that are of interest to us.

Observation 5.1.3. Identity preconditioning is compositional.

Proof. Suppose we have a system with n variables.

The desired linear part of the left model in identity preconditioning is the identity matrix, i.e. it has the form

$$\begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \dots & b_n \end{array} \\ \begin{bmatrix} C_{x_1,b_1} & 0 & \cdots & 0 \\ 0 & C_{x_2,b_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C_{x_n,b_n} \end{bmatrix} \end{array}$$

where $C_{x_i,b_i} = 1$ for $1 \leq i \leq n$.

We can see that for the variable x_i only the coefficient C_{x,b_i} is non zero since we know that $b_i \in \vec{b}_{x_i}$ we can conclude that identity preconditioning is compositional. □

[6] Identity preconditioning is an example of a compositional method where we know the row C_x^{-1}.

Algorithm 11: Compositional preconditioning

input : reference to the component A.

precondition : component B uses the field $B.\vec{\mathcal{U}}$ to store the left model used for input in preconditioning for variables $\vec{x}_B$, similarly, right model is stored in the field $\vec{\mathcal{V}}$.

postcondition: left model for variables $\vec{x}_A$ after preconditioning stored in $A.\vec{\mathcal{U}}'$, right model for variables $\vec{x}_A$ after preconditioning stored in $A.\vec{\mathcal{V}}'$.

1 **Function** *CompPrecondition(A)*
2 $\vec{C}_A \leftarrow$ `DesiredLinearPart`(A) ;
3 **foreach** $x \in \vec{x}_A$ **do**
4 $\mathcal{U}'_x \leftarrow (\lambda\vec{b}.\ c_x + \vec{C}_x\vec{b} + [0,0])|_{\vec{b}_x}$;
5 $\mathcal{V}'_x \leftarrow 0$;
6 **foreach** $z \in \vec{z}_x$ **do**
7 **if** $z \in \vec{x}_A$ **then** `/* z is in component A */`
8 $\mathcal{T} \leftarrow$ `NonConstantPart`(CompLookup$[z].\mathcal{U}_z$);
9 **else** `/* z is solved beforehand, need to extend */`
10 $\mathcal{T} \leftarrow$ `NonConstantPart`(CompLookup$[z].\mathcal{U}_z|^{\vec{b}_A}$)
11 **end**
12 $\mathcal{V}'_x \leftarrow \mathcal{V}'_x + \vec{C}_A^{-1}[x][pv(z)]\mathcal{T}$;
13 **end**
14 **end**
15 **foreach** $b \in \vec{b}_x$ **do**
16 $z \leftarrow pv^{-1}(b)$;
17 **if** $z \in \vec{x}_A$ **then** `/* z is in component A */`
18 $\mathcal{V}'_x \leftarrow \mathcal{V}'_x\{b \mapsto$ CompLookup$[z].\mathcal{V}_z\}$;
19 **else** `/* z is solved beforehand, need to extend */`
20 $\mathcal{V}'_x \leftarrow \mathcal{V}'_x\{b \mapsto$ CompLookup$[z].\mathcal{V}_z|^{\vec{a}_x}\}$;
21 **end**
22 **end**
23 $A.\vec{\mathcal{U}}' \leftarrow \vec{\mathcal{U}}'_A$;
24 $A.\vec{\mathcal{V}}' \leftarrow \vec{\mathcal{V}}'_A$;

Algorithm 12: Computing flow with compositional integration and compositional preconditioning

input : reference to the component A, order k.

precondition : for any component B, left model before preconditioning stored in the field $B.\vec{\mathcal{U}}$, right model before preconditioning stored in the field $B.\vec{\mathcal{V}}$.

postcondition: for component A, left model after preconditioning and integrating stored in the field $A.\vec{\mathcal{U}}^*$, right model after preconditioning stored in the field $A.\vec{\mathcal{V}}'$.

1 **Function** *FullyCompPrecondIntegrate (A, k)*

```
// desired linear part for this component
```
2 `CompPrecondition` (A) ;

```
// integrate the component
```
3 `CompPrecondIntegrate` $(A, A.\vec{\mathcal{U}}', k)$;

4 $A.\vec{\mathcal{U}}^* \leftarrow A.\mathsf{flow}$;

Observation 5.1.4. Parallelepiped preconditioning is compositional.

Proof. Suppose we have a system with n variables.

Let the input to preconditioning be factored model $\vec{\mathcal{U}} \circ \vec{\mathcal{V}}$ with the linear part of the left model $\vec{\mathcal{U}}$ having the form

$$\begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \dots & b_n \end{array} \\ \begin{bmatrix} l_{x_1,b_1} & l_{x_1,b_2} & \cdots & l_{x_1,b_n} \\ l_{x_2,b_1} & l_{x_2,b_2} & \cdots & l_{x_2,b_n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{x_n,b_1} & l_{x_n,b_2} & \cdots & l_{x_n,b_n} \end{bmatrix} \end{array}.$$

Let us inspect the row corresponding to variable x in this matrix.

$$\begin{array}{c} \\ x \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \dots & b_n \end{array} \\ \begin{bmatrix} l_{x,b_1} & l_{x,b_2} & \cdots & l_{x,b_n} \end{bmatrix} \end{array}$$

From the definition of $\vec{a}_x$, we know that if the coefficient of l_{x,b_j} is non zero then $b_j \in \vec{b}_x$. In parallelepiped preconditioning, the matrix corresponding to the desired linear part C is the same as the matrix we inspect, it has to also satisfy this property and we can conclude the parallelepiped preconditioning is compositional. □

Algorithm 13: Compositional integration and compositional preconditioning over multiple time steps

input : sorted list of components $\vec{A}$, factored initial conditions $(\vec{\mathcal{U}}^0, \vec{\mathcal{V}}^0)$ for all variables, order k, time step size Δ.

output: list of factored k-order flowpipes for each time step

Function *FullyCompPrecondSolve($\vec{A}$, $(\vec{\mathcal{U}}^0, \vec{\mathcal{V}}^0)$, k, Δ)*
- flowpipes $\leftarrow [\,]$;
- **foreach** $A \in \vec{A}$ **do**
  - **foreach** $x \in \vec{x}_A$ **do**
    - $A.\mathcal{U}_x \leftarrow \mathcal{U}_x^0|_{\vec{b}_x}$;
    - $A.\mathcal{V}_x \leftarrow \mathcal{V}_x^0|_{\vec{a}_x}$;
  - **end**
- **end**
- **for** $i \leftarrow 0$ **to** $\lceil \frac{t_{final}}{\Delta} \rceil$ **do**
  - $(\vec{\mathcal{U}}^{step}, \vec{\mathcal{V}}^{step}) \leftarrow (0,0)$;
  - **foreach** $A \in \vec{A}$ **do**
    - `FullyCompPrecondIntegrate`(A,k);
    - `// gather flow`
    - **foreach** $x \in \vec{x}_A$ **do**
      - $(\mathcal{U}_x^{step}, \mathcal{V}_x^{step}) \leftarrow (A.\mathcal{U}_x^*|^{\vec{a}}, A.\mathcal{V}_x'|^{\vec{a}})$;
    - **end**
    - `// next initial set`
    - $A.\vec{\mathcal{U}} \leftarrow A.\vec{\mathcal{U}}^*|_{t=\Delta}$;
    - $A.\vec{\mathcal{V}} \leftarrow A.\vec{\mathcal{V}}'$;
  - **end**
  - flowpipes $\leftarrow$ flowpipes $+ [(\vec{\mathcal{U}}^{step}, \vec{\mathcal{V}}^{step})]$;
- **end**
- **return** flowpipes;

Observation 5.1.5. QR preconditioning is non-compositional.

We justify Observation 5.1.5 with the following example.

Example 18. Suppose we have a system with two variables x_1 and x_2 where x_1 is the influencer of x_2, but x_2 is not the influencers of x_1 then we get that $\vec{b}_{x_1} = \{b_1\}$ and $\vec{b}_{x_2} = \{b_1, b_2\}$.

Suppose that the linear part of the left model to preconditioning is

$$\begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cc} b_1 & b_2 \end{array} \\ \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \end{array}.$$

Applying QR decomposition we get the Q matrix as

$$Q = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

In QR preconditioning method, C is equal to Q, for clarity let us annotate this matrix with variables and parameters

$$\begin{array}{c} \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{cc} b_1 & b_2 \end{array} \\ \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \end{array}$$

we can see that the coefficient C_{x_1,b_2} is non-zero here which violates the required property of compositional preconditioning methods.

5.2 Compositional Shrink Wrapping

As preconditioning, shrink wrapping was also created to reduce the overestimation from the remainder interval. We will explore shrink wrapping in the compositional setting next.

We remind that shrink wrapping is making the Taylor models more favourable by absorbing the interval remainders into the polynomial part of the Taylor model. This is done by modifying the coefficients of the terms in the polynomial.

The shrink wrapping method consists of the shrink wrapping phase and the integration phase. Let us first focus on the wrapping phase. It turns out that it is not possible to use composition in the widest sense[7] in this phase.

We will now elaborate on why it is not possible.

Suppose we have a system with a variable x. A key thing in the absorption of the remainder term is determining how far does adding the remainder term extend the set given by the polynomial part of the Taylor model. That is, determining the value

$$\sup_{\vec{a}_1-\vec{a}_0\in\mathbf{i}} p(\vec{a}_1)-p(\vec{a}_0) \tag{5.6}$$

for a Taylor model $(p,\mathbf{i})$ over a domain D.

Suppose that x is influencing some variables. These variables share a parameter with x which means the Taylor model of those variables affects the value (5.6). This implies that we need to take into account the whole system when applying shrink wrapping.

It is, however, possible to use shrink wrapping in the integration phase. In the naive method, we showed that the Picard operator (and therefore integration) can guarantee that Taylor models for some variables will never have some parameters. The same thing holds true for shrink wrapping since it modifies the Taylor model by modifying the scalars of the terms in polynomial and will not introduce new parameters.

We can, therefore, have a scheme where we restrict the parameters before the integration phase to use composition during integration and extend the Taylor models to the non-compositional system during shrink wrapping. We give the algorithm for this in Algorithm 14. The algorithm assumes the existence of function `ShinkWrap` which takes an argument of a Taylor model and returns a Taylor model corresponding to the result of shrink wrapping its argument. The algorithm also uses function `CompNaiveIntegrate` given in Algorithm 7.

For non-compositional shrink wrapping and compositional integration, we give the pseudo-code for integration over multiple time steps in Algorithm 15.

We also mention that it is possible to apply shrink wrapping to a single component separately, the problem with that would be the shrink wrapping factor computed might be invalidated when shrink wrapping later components.

[7] It would be possible to use the fact that in compositional systems the linear part of the flow has a particular structure. This structure will result in a particular matrix which inverse is needed and this inverse could be computed in a compositional manner. However, this computing this inverse is not the dominating part of the computations.

Algorithm 14: Computing flow with compositional integration with non-compositional preconditioning.

input : sorted list of components $\vec{A}$, initial conditions $\vec{\mathcal{T}}^0$ for all variables, order k.

output: flow for all of the variables with k-order Taylor model arithmetic.

Function *CompSWIntegrate($\vec{A}$, $\vec{\mathcal{T}}^0$, k)*

```
// shrink wrap the initial condition
```
$\vec{\mathcal{T}}' \leftarrow$ `ShrinkWrap`$(\vec{\mathcal{T}}^0)$;
```
// integrate the left model
```
foreach $A \in \vec{A}$ **do**
```
// use naive method integration
```
`CompNaiveIntegrate` $(A, \vec{\mathcal{T}}', k)$;
```
// extend the flow for each variable in component
```
foreach $x \in \vec{x}_A$ **do**
$\mathcal{T}_x^* \leftarrow A.\mathsf{flow}_x|^{t,\vec{a}}$;
end
end
return $\vec{\mathcal{T}}^*$

Algorithm 15: Non-compositional shrink wrapping and compositional integration over multiple time steps

input : sorted list of components $\vec{A}$, initial conditions $\vec{\mathcal{T}}^0$ for all variables, order k, time step size Δ.

output: list of k-order flowpipes for each time step

Function *CompSWSolve($\vec{A}$, $\vec{\mathcal{T}}^0$, k, Δ)*

flowpipes $\leftarrow [\,]$;
for $i \leftarrow 0$ **to** $\lceil \frac{t_{final}}{\Delta} \rceil$ **do**
$\vec{\mathcal{T}}^{step} \leftarrow$ `CompSWIntegrate`$(A, \vec{\mathcal{T}}^0, k)$;
flowpipes $\leftarrow$ flowpipes $+ [\vec{\mathcal{T}}^{step}]$;
```
// next initial condition
```
$\vec{\mathcal{T}}^0 \leftarrow \vec{\mathcal{T}}^{step}|_{t=\Delta}$;
end
return flowpipes;

Another problematic aspect of shrink wrapping for us is the need to find a rectangular interval box inside of the range of the Taylor model. The method of doing this and the issue mentioned before are dependent on the coordinate transformation that would result in the Taylor model where the linear part is the unit matrix.

3 cases make getting the unit matrix for the linear part problematic:

1. the linear part is not invertible,
2. more parameters than variables,
3. fewer parameters than variables.

Issue 1 is inherited from the shrink wrapping method in general, we refer to [MB11] for techniques for dealing with it.

Issues 2 and 3 are related to allowing unrestricted Taylor model to represent the initial conditions. Issue 2 could be solved by introducing extra variables. These variables should only be used to compute the shrink wrapping factor and be discarded after doing that. There should be that many variables introduced that the number of variables is equal to the number of parameters. The Taylor models for those variables can be of any form as long as the linear part of the whole system is invertible. In essence, this will result looking at the set in a higher-dimensional space. The set in the higher-dimensional space can be shrink wrapped as usual. This would likely result in a larger shrink wrapping factor than the optimal one (but this can be controlled to a degree by using suitable Taylor models which only have linear polynomials and no remainders).

We do not have a solution to issue 3 and see it as a limitation.

Chapter 6

The Tool CFlow*

To assess the performance of our approach, we have implemented a prototype by extending the (non-compositional) Flow* [CÁS12] version 2.0.0 with our compositional solving method.

This chapter gives a brief overview of our tool CFlow* and Flow*.

6.1 Overview

Flow* is a reachability analyser for non-linear hybrid systems. It is implemented in C++ and based on open source libraries such as the GNU MPFR Library and the GNU Scientific Library. The main functionality of the tool is to solve a hybrid reachability problem. To do that it also needs to provide a validated ODE solver.

To implement our compositional validated ODE solver, we use Flow* as a library that provides modules for interval arithmetic, Taylor model arithmetic and a parser for continuous systems. We list the differences between our ODE solver and the ODE solver from Flow* below when appropriate.

Like Flow* our tool computes the flows of all ODEs for the specified amount of time or until a suitable remainder cannot be found in the validation step. In addition to the compositional setting, we have also an implementation for the non-compositional setting. We have implemented integration without any processing between time steps, integration with shrink wrapping and integration with preconditioning (identity, parallelepiped and QR).

6.2 Arithmetic and Data Types

In Flow*, the library of interval arithmetic is implemented based on the GNU MPFR Library. We refer to [Che15] for more details but mention that all of the round-off errors are taken into account during the computations and real numbers are treated as intervals. This has the consequence that all polynomials in Flow* have interval coefficients.

The data structures of interest to us from Flow* are for representing intervals, monomials, polynomials, Horner forms and Taylor models. Intervals are pairs of two `mpfr_t` type objects[1], monomials are represented as a list of powers of the parameters, polynomials are represented as a list of monomials and Taylor models are pairs that consist of a polynomial and an interval[2].

The key operations in the compositional approach are restricting and extending the sets of parameters used in Taylor models. We have augmented the Flow* library with extra functionality for these operations (all of the above mentioned data structures besides intervals need this extra functionality). Restricting and extending come with a computational cost. The need for this extra functionality arises from the fact that monomials are using a dense representation. This extra cost could be avoided by switching to a sparse representation (monomials as sequences of pairs of parameter names and positive exponents). We estimate the performance benefit from a dense representation well outweighs this cost.

Flow* also includes some techniques for making Taylor model operations more manageable or to improve the accuracy of the arithmetic. We list and briefly describe them:

- **Taylor model simplification**. An order k polynomial with n parameters may contain $\binom{n+k}{k}$ terms. It is common in practical problems that the Taylor models describing the dynamics are such that some terms in the polynomial have tiny coefficients. These terms mostly have a trivial role in the dynamics of the system. To improve the performance, these terms are pushed into the remainder interval by bounding them with intervals which are then subsequently added to the remainder interval. This results in a Taylor model is still a correct overapproximation of the solution but has fewer terms in the polynomial part and has a slightly larger interval remainder.

[1] From GNU MPFR Library.

[2] We assume that the domain of the Taylor model is the unit box.

- **Efficient Taylor model substitution**. A core operation in the Picard operator and preconditioning is substituting a variable in a polynomial with a Taylor model. It is possible to reduce both the number of multiplications and the effect of dependency problem by transforming the polynomial into Horner form before the substitution [CK04].

- **Fast remainder refinement**. Applying the Picard operator in the remainder refinement phase produces Taylor models with the same polynomial part. This is due to the effect that the newly computed terms are pushed to the remainder part due to the use of bounded order Taylor model arithmetic. The Picard operator can be made faster by caching the interval boxes of the terms pushed to the reminder (so that these cached values could be used in the next refinement step instead of computing them again).

  Our tool makes use of all the listed techniques and adapts them to compositional settings.

6.3 Syntax

Our tool uses largely the same syntax as Flow* to describe the systems.

6.3.1 EIVP Description

An EIVP is described with a model file that has the syntax shown in Listing 6.1

Listing 6.1: EIVP syntax

```
continuous reachability
{
  state var <vars>
  setting { <settings> }
  poly ode 1 { <odes>  }
  init { <init> }
}
```

where <vars> is a list of variables in the system, <settings> are the setting used in computing the flow, <odes> is the description of the vector field of EIVP and <init> is the initial condition of the EIVP.

In <vars> is the list of variables expressed with BNF as

```
<vars> ::= x | x, <vars>
```

where x is a variable name.

In <settings> the settings are given by listing them using the defined keywords sometimes together with the parameter.

In <odes> the vector field is given by n equations of the form x′ = ϕ where x is a variable and ϕ is an expression expressed with BNF as

```
ϕ ::= ϕ + ϕ | ϕ - ϕ | ϕ * ϕ | -ϕ | (ϕ) | ϕ^n | x | r
```

where n is a natural number, x is a variable, r is a rational number and s is an alphanumeric literal.

In <init> the initial conditions for all the variables are given by listing variables in predicate `x in [r1,r2]`, where x is a variable and r1 and r2 are rational numbers, such that r1 $\leq$ r2.

6.3.1.1 Settings

We now list the settings that we are using in CFlow*. In the following r is a positive rational number and n is a positive natural number.

- **Time step size**. The duration of the single time step is set to be r with

  ```
  fixed steps r
  ```

- **Integration time**. The duration of the integration is set to be r with

  ```
  time r
  ```

- **Initial remainder estimation**. The initial remainder estimation is set to be r with

  ```
  remainder estimation r
  ```

- **Processing between time steps**. The processing method between time steps is set to be either identity preconditioning, fully compositional identity preconditioning method, QR preconditioning, shrink wrapping after remainder exceeds bounds r or shrink wrapping after every n steps with

  ```
  identity preconditioning |
  compositional identity preconditioning |
  parallelepiped  preconditioning |
  ```

```
QR preconditioning |
shrink wrapping r |
shrink wrapping n
```

All of these preconditioning methods use left model compositional preconditioning except for `QR preconditioning` (which uses non-compositional preconditioning) and `compositional identity preconditioning` (which uses fully compositional preconditioning).

- **Order**. The orders of Taylor models is set to be `n` with

  ```
  fixed orders n
  ```

- **Cutoff**. The width limit used in the Taylor model simplification is set to be `r` with

  ```
  cutoff r
  ```

- **Precision of `mpfr_t`**. The precision used by MPFR library is set to be `n` with

  ```
  precision n
  ```

- **Output prefix**. The prefix of the output filenames is set to be `s` with

  ```
  output s
  ```

- **Decomposition of system**. The compositional methods can be picked by partitioning the variables of the system into components with

  ```
  decomposition [ψ]
  ```

  where ψ denote component by listing. ψ is expressed with BNF as

  ```
  [<vars>] | ψ
  ```

  where `<vars>` uses the above definition.

  Alternatively, non-compositional methods can be picked with

  ```
  no decomposition
  ```

6.3.1.2 Differences from Flow*

In addition to using composition when solving a problem, our tool differs from Flow* in a couple of aspects.

First, in addition to identity and QR preconditioning, our tool also has parallelepiped preconditioning.

Second, in addition to preconditioning our tool has implementation for shrink wrapping as a processing method between the integration of different time steps.

Third, in addition to fixed time step size and fixed Taylor model order, Flow* also has adaptive time step size and adaptive order. Our tool only has implementation for fixed versions of these settings[3].

[3] We have not implemented these additional settings since their interaction with composition should be the same as the settings we have implemented.

Chapter 7

Experiments

7.1 Overview

In this chapter, we assess the performance of our tool CFlow*. Our prototype uses the same parameters and data-structures as Flow*. We refer to Chapter 6 for a more detailed description of the prototype. All experiments presented here were performed on a workstation running Ubuntu 18.04 with Intel 2.80GHz i7-930 and 12GB memory.

Using the prototype, we compute the flows of all ODEs for the specified amount of time or until a suitable remainder cannot be found in the validation step.

We present two classes of experiments. In the first class, we compare the performance of preconditioning methods by examining (a) how far they manage to integrate the system (or if they managed to complete the integration time goal), and (b) then comparing the widths of the flows. In the second class, we use the compositional approach in solving the system with the aforementioned determined "best" preconditioning method. As described in detail below, if a system is non-compositional by nature, we make it compositional "artificially" by considering a composite system composed of copies of the non-compositional system.

The main focus of our experiments is to compare the compositional approach with the non-compositional one. We duplicate the compositional experiments with alternative implementation for one of the underlying data structures. We also compare shrink wrapping with other processing methods and try to analyse why it performs worse.

7.2 Systems

In this section, we shall present the systems (and parameters) used in our experiments. Some of these systems are biological in their nature, some are artificial and some are from external benchmark library.

As a rule we will not go into much details of the systems, since the systems themselves nor their dynamics is not the main focus of our experiments. However, we will briefly describe and explain the modifications that we made to systems from [Mic15b].

7.2.1 Continuous systems

This section presents continuous systems. These systems originate from [Mic15b] and from non-linear continuous benchmarks section of [HyP19].

7.2.1.1 AND-Gate with high inputs (destiffened)

$$\begin{aligned}
\frac{dx_1}{dt} &= -k^{\rightarrow}_{r_1} x_1 x_3 - k^{\rightarrow}_{r_3} x_1 x_5 + k^{\leftarrow}_{r_1} x_4 \\
\frac{dx_2}{dt} &= -k^{\rightarrow}_{r_2} x_2 x_3 - k^{\rightarrow}_{r_4} x_2 x_4 + k^{\leftarrow}_{r_2} x_5 \\
\frac{dx_3}{dt} &= -k^{\rightarrow}_{r_1} x_1 x_3 - k^{\rightarrow}_{r_2} x_2 x_3 + k^{\leftarrow}_{r_1} x_4 + k^{\leftarrow}_{r_2} x_5 \\
\frac{dx_4}{dt} &= -k^{\rightarrow}_{r_4} x_2 x_4 + k^{\rightarrow}_{r_1} x_1 x_3 - k^{\leftarrow}_{r_1} x_4 \\
\frac{dx_5}{dt} &= -k^{\rightarrow}_{r_3} x_1 x_5 + k^{\rightarrow}_{r_2} x_2 x_3 - k^{\leftarrow}_{r_2} x_5 \\
\frac{dx_6}{dt} &= k^{\rightarrow}_{r_3} x_1 x_5 + k^{\rightarrow}_{r_4} x_2 x_4 \\
\frac{dx_7}{dt} &= k^{\rightarrow}_{r_3} x_1 x_5 + k^{\rightarrow}_{r_4} x_2 x_4
\end{aligned}$$

where $k^{\rightarrow}_{r_1} = 0.0003, k^{\leftarrow}_{r_1} = 0.001, k^{\rightarrow}_{r_2} = 0.0003, k^{\leftarrow}_{r_2} = 0.001, k^{\rightarrow}_{r_3} = 0.03, k^{\rightarrow}_{r_4} = 0.03,$. We will use $x_1 \in [0.99, 1.01], x_2 \in [0.99, 1.01], x_3 = 10, x_4 = x_5 = x_6 = x_7 = 0$ for the initial conditions.

This is a less stiff version of the *Simple: And Circuit* library example. The logic function modelled is that of a 2 input AND gate. In the DSD tool library the reactions and their rates are:

$$\begin{aligned}
r_1 : x_1 + x_3 &\rightleftharpoons x_4 && (k^{\rightarrow}_{r_1} = 0.0003 \leftrightarrow k^{\leftarrow}_{r_1} = 0.1126) \\
r_2 : x_2 + x_3 &\rightleftharpoons x_7 && (k^{\rightarrow}_{r_2} = 0.0003 \leftrightarrow k^{\leftarrow}_{r_2} = 0.1126) \\
r_3 : x_1 + x_5 &\rightarrow x_6 + x_7 && (\rightarrow k^{\rightarrow}_{r_3} = 0.0003) \\
r_4 : x_2 + x_4 &\rightarrow x_6 + x_7 && (\rightarrow k^{\rightarrow}_{r_4} = 0.0003)
\end{aligned}$$

Stiffness is a property of ODE systems that can make their numerical solution challenging. One symptom is system components changing value over very different time scales. In this case the concentrations of species x_4 and x_5 initially rise to a peak value

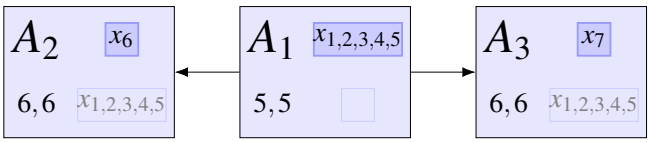


Figure 7.1: Component dependency graph for AND-Gate.

in 15 seconds, but the output species x_7 takes 3×10^5 seconds to rise to 80% of its final value. When attempting validated integrations of this system with Flow*, the computations always diverged by around 10^5 sec. Our de-stiffened version has rates $(0.0003 \leftrightarrow 0.001)$ for the first two reversible reactions and rates $(\rightarrow 0.03)$ for the last two. In the compositional setting we partition this system into 3 components: one with 5 variables and 2 with 1 variable. Component dependency graph for this system is presented in Figure 7.1.

7.2.1.2 AND-OR Gate

This is derived from the *Localization: AND of ORs* example in the DSD tool library. This example models the Boolean function $s \cdot s_1 + s_2 \cdot s_3$. The library version of this example is configured to include *leak* reactions that have reaction rates 5×10^4 times lower than the main reaction rates. These very different rates posed a challenge for Flow*: we made progress only with a very fine time step size and were looking at run-times of over a day. To produce a more straight-forward example we disabled the

leak reactions.

$$
\begin{array}{lcllcl}
\frac{da_1}{dt} & = & -\frac{1}{4}s_2a_1 & \frac{ds_2}{dt} & = & -\frac{1}{4}s_2a_1 \\
\frac{da_0}{dt} & = & -\frac{1}{4}s_0a_0 & \frac{ds_0}{dt} & = & -\frac{1}{4}s_0a_0 \\
\frac{dsp_{12}}{dt} & = & \frac{1}{4}s_2a_1 & \frac{dsp_{13}}{dt} & = & \frac{1}{4}s_2a_1 \\
\frac{dsp_{22}}{dt} & = & \frac{1}{4}s_0a_0 & \frac{dsp_{23}}{dt} & = & \frac{1}{4}s_0a_0 \\
\frac{ds_3}{dt} & = & -\frac{1}{4}s_3sp_{11} & \frac{dsp_{11}}{dt} & = & \frac{1}{4}s_2a_1 - \frac{1}{4}s_3sp_{11} \\
\frac{ds_1}{dt} & = & -\frac{1}{4}s_1sp_{21} & \frac{dsp_{21}}{dt} & = & \frac{1}{4}s_0a_0 - \frac{1}{4}s_1sp_{21} \\
\frac{dsp_{24}}{dt} & = & \frac{1}{4}s_1sp_{21} & \frac{dsp_{25}}{dt} & = & \frac{1}{4}s_1sp_{21} \\
\frac{dsp_{14}}{dt} & = & \frac{1}{4}s_3sp_{11} & \frac{dsp_{15}}{dt} & = & \frac{1}{4}s_3sp_{11} \\
\frac{do_1}{dt} & = & -\frac{1}{4}sp_{16}o_1 & \frac{dsp_{16}}{dt} & = & \frac{1}{4}s_3sp_{11} - \frac{1}{4}sp_{16}o_1 \\
\frac{do_0}{dt} & = & -\frac{1}{4}sp_{26}o_0 & \frac{dsp_{26}}{dt} & = & \frac{1}{4}s_1sp_{21} - \frac{1}{4}sp_{26}o_0 \\
\frac{dsp_{17}}{dt} & = & \frac{1}{4}sp_{16}o_1 & \frac{dsp_{18}}{dt} & = & \frac{1}{4}sp_{16}o_1 \\
\frac{dsp_{27}}{dt} & = & \frac{1}{4}sp_{26}o_0 & \frac{dsp_{28}}{dt} & = & \frac{1}{4}sp_{26}o_0 \\
\frac{dr}{dt} & = & -\frac{1}{4}sp_{19}r - \frac{1}{4}sp_{29}r & \frac{dsp_{19}}{dt} & = & \frac{1}{4}sp_{16}o_1 - \frac{1}{4}sp_{19}r \\
\frac{dsp_{29}}{dt} & = & \frac{1}{4}sp_{26}o_0 - \frac{1}{4}sp_{29}r & \frac{dsp_{10}}{dt} & = & \frac{1}{4}sp_{19}r + \frac{1}{4}sp_{29}r \\
\frac{dsp_{20}}{dt} & = & \frac{1}{4}sp_{19}r & \frac{dsp_{30}}{dt} & = & \frac{1}{4}sp_{29}r
\end{array}
$$

We use initial conditions that set the two inputs of one AND gate at a logic 1 and the two of the other at a logic 0.

In the compositional setting we partition this system into 22 components: one with 3 variables, 6 with 2 variables and 15 with 1 variable. Component dependency graph for this system is presented in Figure 7.2.

7.2.1.3 Brusselator

The Brusselator is a model for autocatalytic reaction. It is described by the following ODEs

$$
\begin{array}{lcl}
\frac{dx}{dt} & = & A + x^2y - Bx - x \\
\frac{dy}{dt} & = & Bx - x^2y
\end{array}
$$

where x and y are the concentration of chemicals involved. A and B are constant parameters (the fixed point of Brusselator becomes unstable when $B > 1 + A^2$). We will use values $A = 1$, $B = \frac{3}{2}$ for parameters and $x \in [0.8, 1]$ and $y \in [0, 0.2]$ for the initial conditions.

We will not present the graphical compositional description of this system due to it only containing a single component.

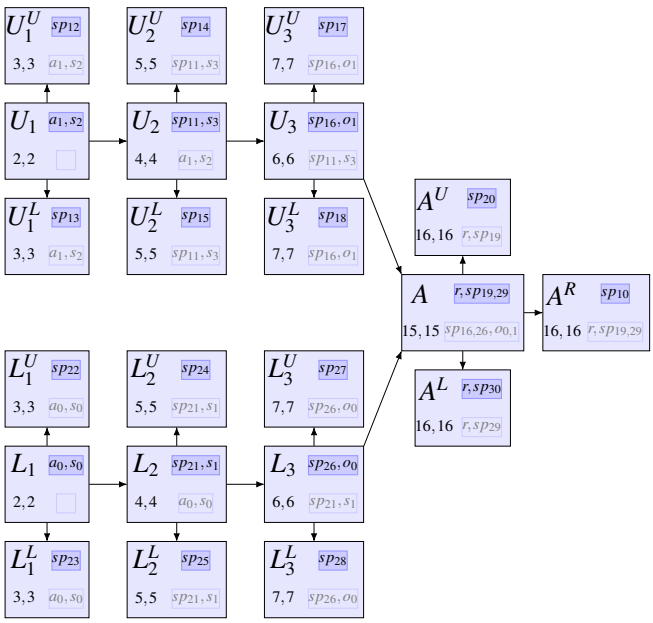


Figure 7.2: Component dependency graph for AND-OR Gate.

7.2.1.4 Buckling column

The buckling column [Zha92] can be described by the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= A + x^2 y - Bx - x \\ \frac{dy}{dt} &= Bx - x^2 y \end{aligned}$$

We will use $x \in [-0.5, -0.4]$ and $y \in [-0.5, -0.4]$ for the initial conditions.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.1.5 Jet engine

The Moore-Greitzer model of a jet engine [APS06] is described by the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= -y - \tfrac{3}{2}x^2 - \tfrac{1}{2}x^3 - \tfrac{1}{2} \\ \frac{dy}{dt} &= 3x - y \end{aligned}$$

We will use the initial conditions $x \in [0.8, 1.2]$ and $y \in [0.8, 1.2]$.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.1.6 Lorentz system

The Lorentz system was developed by Edward Lorenz for modelling atmospheric convection. It is described by the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y-x) \\ \frac{dy}{dt} &= x(\rho-z)-y \\ \frac{dz}{dt} &= xy-\beta z \end{aligned}$$

where $\sigma = 10$, $\rho = \frac{8}{3}$, $\beta = 28$. We will use $x \in [14.999, 15.001]$, $y \in [14.999, 15.001]$ and $z \in [35.999, 36.001]$ for the initial conditions.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.1.7 Lotka-Volterra

Lotka-Volterra equations describe how two species interact if they have a predator-prey relationship. It is described by the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= \alpha x - \beta xy \\ \frac{dy}{dt} &= \delta xy - \gamma y \end{aligned}$$

where x is the population of of prey, y is the population of the predators. α, β, γ and δ are constant positive parameters. We will use values $\alpha = \frac{3}{2}$, $\beta = 1$, $\gamma = 3$, $\delta = 1$ for the parameters. We will use $x \in [4.8, 5.2]$ and $y \in [1.8, 2.2]$ for the initial conditions.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.1.8 Moore's rotational system

Moore illustrates the wrapping effect with his rotational system in [Moo66]. This system is described by the following ODEs

$$\begin{aligned} \frac{dx_1}{dt} &= -x_2 \\ \frac{dx_2}{dt} &= x_1 \end{aligned}$$

We will use $x_1 \in [10, 11]$ and $x_2 \in [-1, 1]$ for the initial conditions.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.1.9 Roessler attractor

The Roessler attractor can be modelled with the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= -y - z \\ \frac{dy}{dt} &= x + ay \\ \frac{dz}{dt} &= b + z(x - c) \end{aligned}$$

where $a = 0.2$, $b = 0.2$, $c = 5.7$. We will use $x \in [-0.2, 0.2]$, $y \in [-8.6, -8.2]$ and $z \in [-0.2, 0.2]$ for the initial conditions.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.1.10 Van der Pol oscillator

Van der Pol oscillator can be modelled with the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= y \\ \frac{dy}{dt} &= \mu(1 - x^2)y - x \end{aligned}$$

where μ is a constant parameter (we will use the typical $\mu = 1$). We will use $x \in [1.25, 1.55]$ and $y \in [2.25, 2.35]$ for the initial conditions.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.2 Artificial systems

This section presents artificially created systems. These systems are made in order to see how composition behaves under some specific type of a system.

7.2.2.1 Linear compositional

We can model a fully compositional system with simple linear dynamics with the following ODEs

$$\frac{dx_i}{dt} = -\frac{1}{10}x_i$$

where the dimension of the system is n (we will specify n for specific experiments).

We will use $x_i \in [0.5, 1.0]$ for $i \in \{1 \dots n\}$ for the initial conditions.

Component dependency graph for this system with 4 dimensions is presented in Figure 7.3.

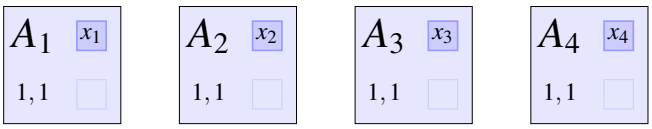


Figure 7.3: Component dependency graph for a 4-dimensional linear compositional system.

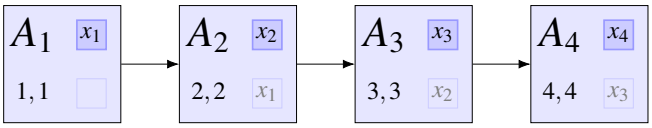


Figure 7.4: Component dependency graph for 4-dimensional linear dependent system.

7.2.2.2 Linear dependent

An artificial system that is partially, naturally compositional is described by the following ODEs

$$\begin{aligned} \frac{dx_1}{dt} &= -\frac{1}{10}x_1 \\ \frac{dx_i}{dt} &= \frac{1}{10}x_{i-1} - \frac{1}{10}x_i \end{aligned}$$

where the dimension of the system is n (we will specify n for specific experiments) and $i \in \{2 \dots n\}$.

We will use this system with the initial conditions $x_j \in [0.5, 1.0]$ for $j \in \{1 \dots n\}$.

In here, each variable is again put into a separate component and all components (except the first) have exactly one component dependency. This system is not generally favourable for the compositional method as the components contain the parameters from all of their ancestor components. The graphical compositional description of this system with 4 dimensions is presented in Figure 7.4.

7.2.2.3 Pairwise dependent

We can model a partially compositional dynamics with the non-linear dynamics involving multiple variables with the following ODEs

$$\begin{aligned} \frac{dx_{2i}}{dt} &= -\frac{1}{10}x_{2i}x_{2i+1} \\ \frac{dx_{2i+1}}{dt} &= -\frac{1}{10}x_{2i}x_{2i+1} \end{aligned}$$

where the dimension of the system is n (we will specify n for specific experiments).

We will use $x_i \in [0.5, 1.0]$ for $i \in \{1 \dots n\}$ for the initial conditions.

Component dependency graph for this system with 4 dimensions is presented in Figure 7.5.

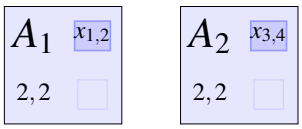


Figure 7.5: Component dependency graph for a 4-dimensional pairwise dependent system.

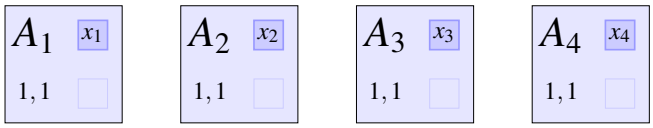


Figure 7.6: Component dependency graph for 4-dimensional squared degradation system.

7.2.2.4 Squared degradation

We can model a fully compositional system with simple non-linear dynamics with the following ODEs

$$\frac{dx_i}{dt} = -\frac{1}{10}x_i^2$$

where the dimension of the system is n (we will specify n for specific experiments).

We will use $x_i \in [0.5, 1.0]$ for $i \in \{1 \dots n\}$ for the initial conditions.

Component dependency graph for this system with 4 dimensions is presented in Figure 7.6.

7.2.3 Initial modes of hybrid systems

This section presents hybrid systems that are made into continuous systems by only viewing the initial mode of them. These systems originate from both linear hybrid benchmarks and non-linear hybrid benchmark sections of [HyP19].

7.2.3.1 Bouncing ball

The initial mode of the hybrid system describing the dynamics of a bouncing ball is modelling the dropping of the ball. It is described by the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= v \\ \frac{dv}{dt} &= -9.81 \end{aligned}$$

We will use the initial conditions $x \in [10, 10.2]$ and $v = 0$.

The graphical compositional description of this system is presented in Figure 7.7.

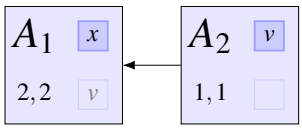


Figure 7.7: Component dependency graph for Bouncing ball system.

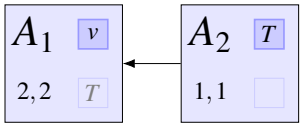


Figure 7.8: Component dependency graph for the Cruise control system.

7.2.3.2 Cruise control

The goal of the the cruise control hybrid system [Oeh11] is to drive the velocity of a vehicle toward a set point defining a target velocity. The initial mode is dealing with breaking and is modelled by the following ODEs

$$\begin{aligned} \frac{dv}{dt} &= -T - \frac{5}{2} \\ \frac{dT}{dt} &= 1 \end{aligned}$$

We will use the initial conditions $v \in [15,40]$ and $T \in [0,2.5]$.

The graphical compositional description of this system is presented in Figure 7.8.

7.2.3.3 Glycemic control

The initial mode of the glycemic control in diabetic patients can be modelled with the following ODEs

$$\begin{aligned} \frac{dG}{dt} &= -p_1 G - X(G+G_B) + g(t) \\ \frac{dX}{dt} &= -p_2 X + p_3 I \\ \frac{dI}{dt} &= -n(I+I_B) + \frac{1}{V_I} i(t) \end{aligned}$$

where G is plasma glucose concentration above the basal value G_B and I is the plasma insulin concentration above the basal value I_B. X is the insulin concentration in an interstitial chamber. p_1, p_2, p_3, V_I, n, G_B and I_b are constant parameters. We will use values $p_1 = 0.01, p_2 = 0.025, p_3 = 0.000013, V_I = 12, n = 0.093, G_B = 4.5, I_b = 15$.

In the initial mode, the influx of glucose $g(t)$ after a meal is modelled as $g(t) = \frac{t}{60}$.

We create two systems based on the insulin control strategies $i(t)$. For first model (named Glycemic control 1) we set it as $i(t) = \frac{25}{3}$ [FKS+85] and for the second model (named Glycemic control 2) we set it as $i(t) = 1 + 0.01852G(t)$ [Fis91].

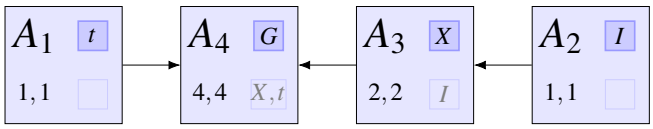


Figure 7.9: Component dependency graph for the Glycemic control 1 system.

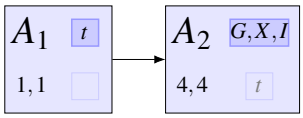


Figure 7.10: Component dependency graph for the Glycemic control 2 system.

The graphical compositional description of the first system is presented in Figure 7.9.

The graphical compositional description of the second system is presented in Figure 7.10.

7.2.3.4 Filtered oscillator

We present 4 versions of the initial mode of the switched oscillator together with filter. In all of these the variables x and y produce the oscillation, z is the output and rest of the variables smooth the output. We group the different oscillators by the dimension of the filter.

The ODEs for the oscillator with 4-dimensional filter are the following

$$\begin{array}{rclrcl} \frac{dx}{dt} & = & -2x+1.4 & \frac{dy}{dt} & = & -y-0.7 \\ \frac{dx_1}{dt} & = & 5x-5x_1 & \frac{dx_2}{dt} & = & 5x_1-5x_2 \\ \frac{dx_3}{dt} & = & 5x_2-5x_3 & \frac{dz}{dt} & = & 5x_3-5z \end{array}$$

The ODEs for the oscillator with 8-dimensional filter are the following

$$\begin{array}{rclrcl} \frac{dx}{dt} & = & -2x+1.4 & \frac{dy}{dt} & = & -y-0.7 \\ \frac{df4aX_1}{dt} & = & 5x-5f4aX_1 & \frac{df4aX_2}{dt} & = & 5f4aX_1-5f4aX_2 \\ \frac{df4aX_3}{dt} & = & 5f4aX_2-5f4aX_3 & \frac{df8X_1}{dt} & = & 5f4aX_3-5f8X_1 \\ \frac{df4bX_1}{dt} & = & 5f8X_1-5f4bX_1 & \frac{df4bX_2}{dt} & = & 5f4bX_1-5f4bX_2 \\ \frac{df4bX_3}{dt} & = & 5f4bX_2-5f4bX_3 & \frac{dz}{dt} & = & 5f4bX_3-5z \end{array}$$

The ODEs for the oscillator with 16-dimensional filter are the following

$$
\begin{aligned}
\frac{dx}{dt} &= -2x+1.4 & \frac{dy}{dt} &= -y-0.7\\
\frac{df8aF4aX_1}{dt} &= 5x-5f8aF4aX_1 & \frac{df8aF4aX_2}{dt} &= 5f8aF4aX_1-5f8aF4aX_2\\
\frac{df8aF4aX_3}{dt} &= 5f8aF4aX_2-5f8aF4aX_3 & \frac{df8aX_1}{dt} &= 5f8aF4aX_3-5f8aX_1\\
\frac{df8aF4bX_1}{dt} &= 5f8aX_1-5f8aF4bX_1 & \frac{df8aF4bX_2}{dt} &= 5f8aF4bX_1-5f8aF4bX_2\\
\frac{df8aF4bX_3}{dt} &= 5f8aF4bX_2-5f8aF4bX_3 & \frac{dX_1}{dt} &= 5f8aF4bX_3-5X_1\\
\frac{df8bF4aX_1}{dt} &= 5X_1-5f8bF4aX_1 & \frac{df8bF4aX_2}{dt} &= 5f8bF4aX_1-5f8bF4aX_2\\
\frac{df8bF4aX_3}{dt} &= 5f8bF4aX_2-5f8bF4aX_3 & \frac{df8bX_1}{dt} &= 5f8bF4aX_3-5f8bX_1\\
\frac{df8bF4bX_1}{dt} &= 5f8bX_1-5f8bF4bX_1 & \frac{df8bF4bX_2}{dt} &= 5f8bF4bX_1-5f8bF4bX_2\\
\frac{df8bF4bX_3}{dt} &= 5f8bF4bX_2-5f8bF4bX_3 & \frac{dz}{dt} &= 5f8bF4bX_3-5z
\end{aligned}
$$

The ODEs for the oscillator with 32-dimensional filter are the following

$$
\begin{aligned}
\frac{dx}{dt} &= -2x+1.4 & \frac{dy}{dt} &= -y-0.7\\
\frac{df8aF4aX_1}{dt} &= 5x-5f8aF4aX_1 & \frac{df8aF4aX_2}{dt} &= 5f8aF4aX_1-5f8aF4aX_2\\
\frac{df8aF4aX_3}{dt} &= 5f8aF4aX_2-5f8aF4aX_3 & \frac{df8aX_1}{dt} &= 5f8aF4aX_3-5f8aX_1\\
\frac{df8aF4bX_1}{dt} &= 5f8aX_1-5f8aF4bX_1 & \frac{df8aF4bX_2}{dt} &= 5f8aF4bX_1-5f8aF4bX_2\\
\frac{df8aF4bX_3}{dt} &= 5f8aF4bX_2-5f8aF4bX_3 & \frac{dX_1}{dt} &= 5f8aF4bX_3-5X_1\\
\frac{df8bF4aX_1}{dt} &= 5X_1-5f8bF4aX_1 & \frac{df8bF4aX_2}{dt} &= 5f8bF4aX_1-5f8bF4aX_2\\
\frac{df8bF4aX_3}{dt} &= 5f8bF4aX_2-5f8bF4aX_3 & \frac{df8bX_1}{dt} &= 5f8bF4aX_3-5f8bX_1\\
\frac{df8bF4bX_1}{dt} &= 5f8bX_1-5f8bF4bX_1 & \frac{df8bF4bX_2}{dt} &= 5f8bF4bX_1-5f8bF4bX_2\\
\frac{df8bF4bX_3}{dt} &= 5f8bF4bX_2-5f8bF4bX_3 & \frac{dX_2}{dt} &= 5f8bF4bX_3-5X_2\\
\frac{df8cF4aX_1}{dt} &= 5X_2-5f8cF4aX_1 & \frac{df8cF4aX_2}{dt} &= 5f8cF4aX_1-5f8cF4aX_2\\
\frac{df8cF4aX_3}{dt} &= 5f8cF4aX_2-5f8cF4aX_3 & \frac{df8cX_1}{dt} &= 5f8cF4aX_3-5f8cX_1\\
\frac{df8cF4bX_1}{dt} &= 5f8cX_1-5f8cF4bX_1 & \frac{df8cF4bX_2}{dt} &= 5f8cF4bX_1-5f8cF4bX_2\\
\frac{df8cF4bX_3}{dt} &= 5f8cF4bX_2-5f8cF4bX_3 & \frac{dX_3}{dt} &= 5f8cF4bX_3-5X_3\\
\frac{df8dF4aX_1}{dt} &= 5X_3-5f8dF4aX_1 & \frac{df8dF4aX_2}{dt} &= 5f8dF4aX_1-5f8dF4aX_2\\
\frac{df8dF4aX_3}{dt} &= 5f8dF4aX_2-5f8dF4aX_3 & \frac{df8dX_1}{dt} &= 5f8dF4aX_3-5f8dX_1\\
\frac{df8dF4bX_1}{dt} &= 5f8dX_1-5f8dF4bX_1 & \frac{df8dF4bX_2}{dt} &= 5f8dF4bX_1-5f8dF4bX_2\\
\frac{df8dF4bX_3}{dt} &= 5f8dF4bX_2-5f8dF4bX_3 & \frac{dz}{dt} &= 5f8dF4bX_3-5z
\end{aligned}
$$

For all of these systems, we will use the initial conditions where every variable is zero, besides $x \in [0.2, 0.3]$ and $y \in [-0.1, 0.1]$.

The graphical compositional description of the oscillator with 4-dimensional filter is presented in Figure 7.11.

The graphical compositional description of the oscillator with 8-dimensional filter is presented in Figure 7.12.

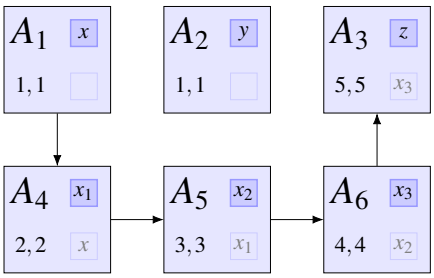


Figure 7.11: Component dependency graph for the oscillator with 4-dimensional filter system.

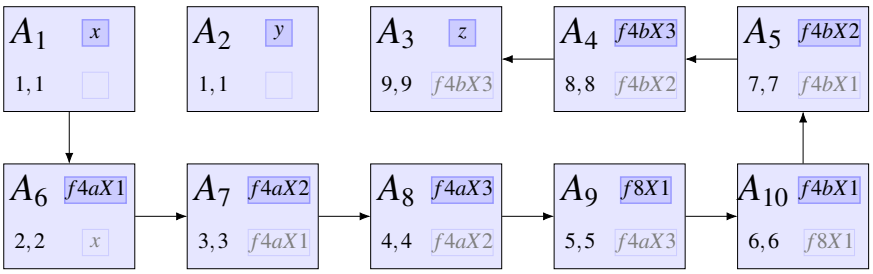


Figure 7.12: Component dependency graph for the oscillator with 8-dimensional filter system.

The graphical compositional description of the oscillator with 16-dimensional or with 32-dimensional filter are analogous to the ones with lower dimensional filters and will not be presented.

7.2.3.5 Non-holonomic integrator

The initial mode for Brockett's non-holonomic integrator can be described by the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= 1 \\ \frac{dy}{dt} &= 1 \\ \frac{dx}{dt} &= x - y \end{aligned}$$

We will use the initial conditions $x = 0$, $y = 0$ and $z \in [14.9, 15.1]$.

The graphical compositional description of this system is presented in Figure 7.13.

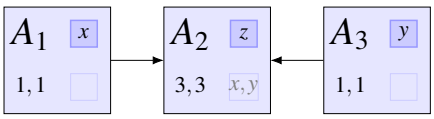


Figure 7.13: Component dependency graph for the non-holonomic integrator system.

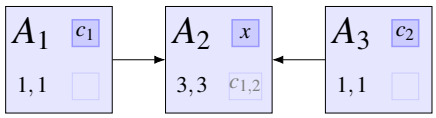


Figure 7.14: Component dependency graph for the rod reactor system.

7.2.3.6 Rod reactor

The initial mode of a system controlling the coolant temperature in the tank using two rods with different cooling dynamics [VVS99] can be modelled with the following ODEs

$$\begin{aligned} \frac{dx}{dt} &= \frac{1}{10}x - 50 \\ \frac{dc_1}{dt} &= 1 \\ \frac{dc_2}{dt} &= 1 \end{aligned}$$

We will use the initial conditions $x \in [510, 520]$, $c_1 = 20$ and $c_2 = 20$.

The graphical compositional description of this system is presented in Figure 7.14.

7.2.3.7 Spiking neurons

The model of spiking neurons can be described with the following ODEs

$$\begin{aligned} \frac{dv}{dt} &= \frac{1}{C}(k(v - v_r)(v - v_t) - u + I) \\ \frac{du}{dt} &= a(b(v - v_r) - u) \end{aligned}$$

We present two models for spiking neurons.

In the first one (named Neuron 1), the constant parameters have the values $C = 100$, $v_r = -60$, $v_t = -40$, $I = 70$, $a = 0.03$ and $b = -2$. The parameter k in the initial model has the value $k = 0.7$. We will use the initial conditions $v \in [-61, -59]$ and $u \in [-1, 1]$.

In the second one (named Neuron 2), the constant parameters have the values $C = 100$, $v_r = -56$, $v_t = -42$, $I = 300$, $a = 0.03$, $b = 8$ and $k = 1$. We will use the initial conditions $v \in [-50.5, -49.5]$ and $u \in [-0.5, 0.5]$.

We will not present the graphical compositional description for these systems due to the fact that they only contain a single component each.

7.2.3.8 Switching system

The initial mode of a 5-dimensional switching linear system can be modelled with the following ODEs

$$\frac{d\vec{x}}{dt} = A\vec{x} + B$$

where

$$A = \begin{bmatrix} -0.8047 & 8.7420 & -2.4591 & -8.2714 & -1.8640 \\ -8.6329 & -0.5860 & -2.1006 & 3.6035 & -1.8423 \\ 2.4511 & 2.2394 & -0.7538 & -3.6934 & 2.4585 \\ 8.3858 & -3.1739 & 3.7822 & -0.6249 & 1.8829 \\ 1.8302 & 1.9869 & -2.4539 & -1.7726 & -0.7911 \end{bmatrix}$$

and

$$B = \begin{bmatrix} [-0.0845, 0.0845] \\ 0 \\ 0 \\ 0 \\ [-0.7342, 0.7342] \end{bmatrix}$$

We will use the initial conditions $x_1 = 3.1$, $x_2 = 4$, $x_3 = 0$, $x_4 = 0$ and $x_5 = 0$.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.3.9 Three vehicle platoon

Collision free navigation of the three vehicle platoon with no communication problems can be modelled with the following ODEs

$$\frac{d\vec{x}}{dt} = A\vec{x} + Ba_L$$

where A is the matrix

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.6050 & 4.8680 & -3.5754 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.1936 & 3.6258 & -3.2396 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0.7132 & 3.5730 & -0.0964 & 0.8472 & 3.2568 & -0.0876 & 1.2726 & 3.0720 & -3.1356 \end{bmatrix}$$

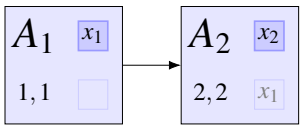


Figure 7.15: Component dependency graph for the two tanks system.

and

$$B = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

where in the state vector $\vec{x} = [e_1, \dot{e_1}, a_1, e_2, \dot{e_2}, a_2, e_3, \dot{e_3}, a_3]$, a_i is the acceleration of vehicle i, e_i is the difference between the distance d_i of the truck i to its predecessor and a reference distance $d_{ref,i}$ (defined as $e_i = d_i - d_{ref,i}$), $\dot{e_i}$ is the derivative of e_i and a_L is the acceleration of the leader.

We will use the initial conditions $e_1 \in [0.9, 1.1]$, $\dot{e_1} \in [0.9, 1.1]$, $a_1 \in [0.9, 1.1]$, $e_2 \in [0.9, 1.1]$, $\dot{e_2} \in [0.9, 1.1]$, $a_2 \in [0.9, 1.1]$, $e_3 \in [0.9, 1.1]$, $\dot{e_3} \in [0.9, 1.1]$ and $a_3 \in [0.9, 1.1]$.

We will not present the graphical compositional description of this system due to it only containing a single component.

7.2.3.10 Two tanks

The initial mode of a two tank models the dynamics where the first tank has an outside inflow source, there is a drain between the tanks and the second tank has both a constant outflow and a controlled outflow can be modelled with the following ODEs

$$\begin{aligned} \frac{dx_1}{dt} &= -x_1 - 2 + [-0.01, 0.01] \\ \frac{dx_2}{dt} &= x_1 - x_2 - 5 + [0.01, 0.01] \end{aligned}$$

We will use the initial conditions $x_1 \in [1.5, 2.5]$ and $x_2 = 1$.

The graphical compositional description of this system is presented in Figure 7.15.

7.3 Compositional Processing Method Experiments

We will apply different compositional processing methods to above-listed systems using our tool CFlow*. The computed flows for each variable are Taylor models, and we can thus illustrate them by bounding them with interval boxes over time (cf. Figure 7.16 or Section B.1). Our goal is to compute flows until a specified integration time is reached. Sometimes that is not possible because the remainder interval used in val-

System	Time Goal	Time step	Order	Cutoff
AND-Gate	1000	10	5	10^{-15}
AND-OR Gate	40	0.05	2	10^{-15}
Brusselator	15	0.03	3	10^{-12}
Buckling column	10	0.01	4	10^{-12}
Jet engine	10	0.03	4	10^{-12}
Lorentz system	6.5	0.003	4	10^{-10}
Lotka-Volterra	10	0.02	4	10^{-20}
Moore's rotational system	10	0.1	5	10^{-15}
Rossler attractor	6	0.02	6	10^{-12}
Van der Pol oscillator	7	0.02	5	10^{-12}
Bouncing ball	10	0.1	5	10^{-15}
Cruise control	100	0.1	5	10^{-15}
Glycemic control (1, 2)	360	0.1	2	10^{-10}
Filtered oscillator (4, 8, 16, 32)	4	0.05	8	10^{-15}
Neuron 1	1000	0.02	4	10^{-12}
Neuron 2	200	0.02	4	10^{-12}
Non-holonomic integrator	7.5	0.01	5	10^{-12}
Rod reactor	50	0.1	5	10^{-15}
Switching system	0.1	0.01	15	10^{-15}
Two tanks	2	0.01	10	10^{-15}
Three vehicle platoon	12	0.02	10	10^{-15}
Linear compositional	1	0.1	5	10^{-12}
Linear dependent	1	0.1	5	10^{-12}
Pairwise dependent	1	0.1	5	10^{-12}
Squared degradation	100	1	5	10^{-15}

Table 7.1: Model parameter values of the systems.

idating the existence and uniqueness cannot be computed[1]. If that is the case then our tool halts, reporting that it can only compute the flow up to the given time.

The following experiments are concerned with determining the most suitable processing method for a given system. Given two processing methods A and B, we say that A is *better* than B if (i) A integrates further than B, or (ii) A and B integrate for the same amount of time but A produces a flow that can be bounded by a smaller interval box. For example, consider different processing methods with the jet engine model. The data for this comparison using identity preconditioning, parallelepiped preconditioning, QR preconditioning, no processing, shrink wrapping at every time step, shrink wrapping at every second time step, shrink wrapping at every fifth time step and shrink wrapping at every tenth time step is given in Figure 7.16. We can see that for this system parallelepiped method manages to compute flowpipes for a considerably longer time period, and is thus deemed *better* for this problem. Assuming that parallelepiped preconditioning managed to integrate only as far as QR preconditioning, then we would again prefer parallelepiped preconditioning as the flowpipe can be bounded by a much smaller interval box.

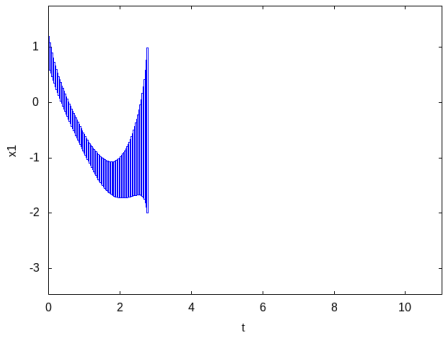
Not all processing methods can be used together with preconditioning and composition. Out of the methods presented in this thesis only identity preconditioning, parallelepiped preconditioning or no processing at all are suitable. We present the data for these methods in Table 7.2[2] (we present the data for other processing methods in Section B.2).

One some systems processing method does not make a difference. If that is the case, then we will prefer identity preconditioning because we can experiment with different levels of composition on that method. Based on Table 7.2, we will prefer

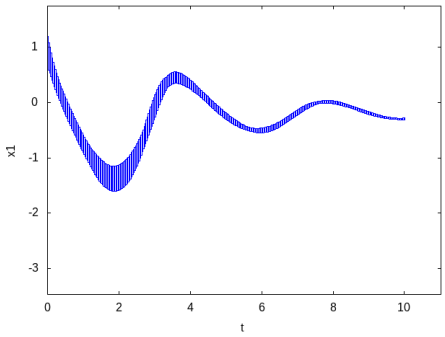
- no processing on the systems Buckling column, Van der Pol oscillator and Three vehicle platoon;
- parallelepiped preconditioning on the systems Jet engine, Lotka-Volterra and Moore's rotational;
- identity preconditioning with all other systems.

[1] Either because the remainder would be too large or that matrices are nonsingular (in parallelepiped preconditioning).

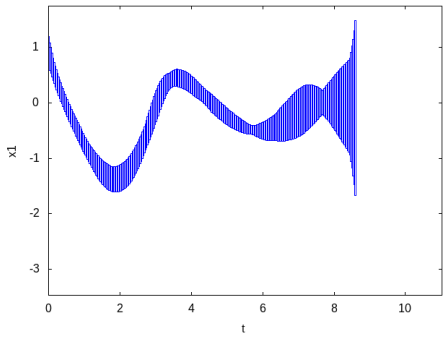
[2] We are using 2-dimensional systems for the artificially created systems



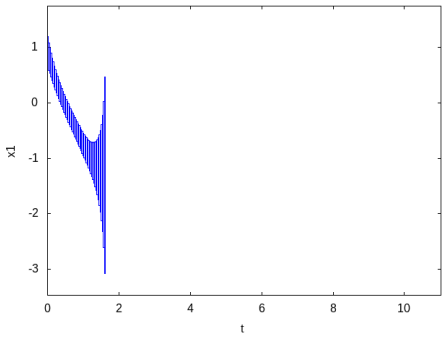
(a) Identity preconditioning



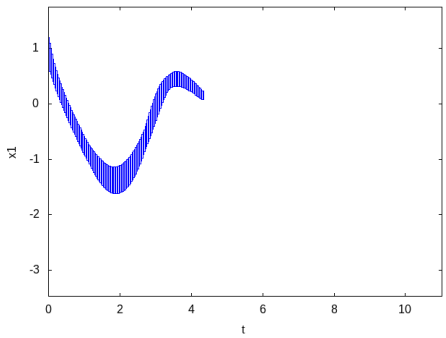
(b) Parallelepiped preconditioning



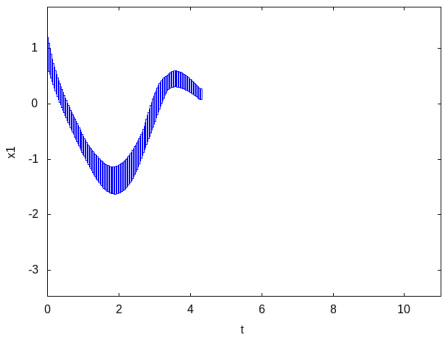
(c) QR preconditioning



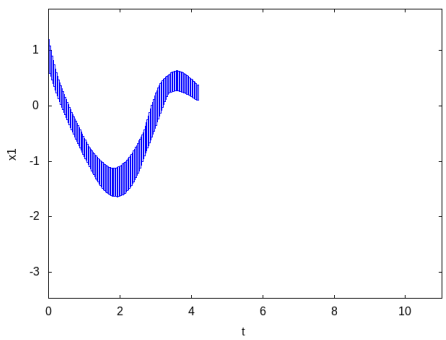
(d) No processing



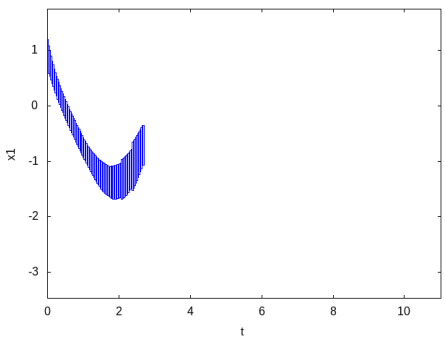
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps



(g) Shrink wrapping after 5 time steps



(h) Shrink wrapping after 10 time steps

Figure 7.16: Flowpipes for Jet engine system using different processing methods.

Example	Processing method					
	ID		PA		Non	
	IP	Wid	IP	Wid	IP	Wid
AND-Gate	**1000**	**0.00676471**	0	–	570	MAX
AND-OR Gate	**40**	**0.0243657**	14.55	0.0324645	40	0.117947
Brusselator	**6**	**MAX**	2.4	MAX	1.89	MAX
Buckling col	7.15	MAX	4.82	MAX	**10**	**2.29083**
Jet engine	2.91	MAX	**10.02**	**0.030186**	1.65	MAX
Lorentz	**1.725**	**MAX**	0.627	MAX	0.912	MAX
Lotka-Volterra	3.3	MAX	**10**	**2.03082**	2.14	MAX
Moore rot	10	2.53292	**10**	**2.44783**	10	2.79329
Roessler	**6**	**1.34766**	0	–	6	4.88364
Vanderpol	7	0.555849	2.86	MAX	**7**	**0.234957**
Linear	**1**	**0.466105**	**1**	**0.466105**	**1**	**0.466105**
Lin-dep	**1**	**0.466105**	**1**	**0.466105**	**1**	**0.466105**
Sqr-deg	**100**	**0.0109325**	**100**	**0.0109325**	100	0.314886
Pairwise	**60**	**0.117493**	45	MAX	60	0.195796
Bouncing ball	**10**	**9.96095**	0	–	**10**	**9.96095**
Cruise control	**100.1**	**285.505**	**100.1**	**285.505**	**100.1**	**285.505**
Glycemic 1	**360**	**0.623517**	0	–	2.9	MAX
Glycemic 2	**360**	**0.201601**	0	–	2.9	MAX
Filtered osc 4	**4**	**5.56115e-05**	0	–	4	5.56167e-05
Filtered osc 8	**4**	**5.56115e-05**	0	–	4	5.56167e-05
Filtered osc 16	**4**	**5.56115e-05**	0	–	4	5.56167e-05
Filtered osc 32	**4**	**5.56115e-05**	0	–	4	5.56167e-05
Neuron 1	**98.78**	**MAX**	78.68	MAX	23.6	MAX
Neuron 2	**10.24**	**MAX**	10.2	MAX	10.22	MAX
Non-holonomic	**7.5**	**0.01**	0	–	**7.5**	**0.01**
Rod reactor	**50**	**MAX**	0	–	**50**	**MAX**
Switching	**0.1**	**0.255531**	0	–	0.1	0.279511
Two tanks	**2**	**0.160753**	0	–	2	0.272389
Three vehicle	12	0.0233121	12	0.0233117	**12**	**0.0233116**

Table 7.2: Experimental results for compositional processing methods.

Abbreviations: Id : identity preconditioning, Pa : parallelepiped preconditioning, Non : no processing, IP: integration progress, Wid: wid of the flowpipe at max IP.

7.4 Compositional vs Non-Compositional

This section is concerned with comparing the compositional and non-compositional approaches.

We use the systems as defined in Section 7.2. The artificial systems are used with no modifications. The continuous and initial modes of hybrid systems are in general not compositional and we will make them compositional by creating a bigger system containing 10 identical copies of the original system. Some of the initial modes of hybrid systems had some degree of composition naturally, in addition to the experiments with 10 copies, we will also experiment with them with no such copying.

In all of the systems, we only use one processing method, which we pick as determined to be the best processing method in the previous section.

Since our compositional flow is practically identical to the non-compositional one, we refrain from presenting the flow itself, but report instead the time needed to compute it and the time spent in key parts of the algorithm. These times correspond to:

- total time needed to solve the system (total),
- total time spent in the integration phase (total integration),
- computing the polynomial in the Picard iteration (polynomial),
- validating the existence and uniqueness of the solution (validating),
- tightening the remainder interval (refinement),
- mapping before preconditioning (mapping 1, mostly extending the domain of Taylor models),
- preconditioning the left and right model (preconditioning),
- mapping after preconditioning (mapping 2, mostly restricting the domain of Taylor models).

Tables 7.3, 7.5, 7.6 and 7.8 present the performance data gathered from experiments. We present this data slightly differently also in Tables 7.4, 7.7 and 7.9 where we present the time spent in the compositional methods as the percentages of the time spent in non-compositional method.

Example	Proc	Comp	Times							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
AND-Gate	ID	FC	33.880	27.196	12.884	13.985	0.122	0.056	6.410	–
		LC	34.541	27.698	13.002	14.357	0.127	0.109	6.617	0.011
		NC	27.596	20.928	6.666	14.007	0.225	0.101	6.393	0.003
AND-OR Gate	ID	FC	21.441	12.319	3.339	6.465	1.269	1.350	6.681	–
		LC	27.394	13.896	3.442	7.413	1.285	0.720	11.884	0.285
		NC	46.196	33.702	4.341	12.555	16.615	0.502	10.912	0.159
Brusselator	ID	FC	6.648	4.325	1.263	2.762	0.256	0.022	2.134	–
		LC	8.156	4.542	1.301	2.824	0.259	0.143	3.314	0.018
		NC	11.580	7.829	2.020	3.962	1.813	0.126	3.233	0.028
Buckling col	NO	FC	85.082	83.751	34.333	47.831	0.451	–	–	–
		NC	115.316	113.946	41.528	68.455	3.080	–	–	–
Jet engine	PA	LC	41.602	15.473	5.427	9.250	0.492	0.424	25.254	0.054
		NC	47.418	21.446	7.058	10.777	3.474	0.354	24.925	0.068
Lorentz	ID	FC	46.336	25.658	9.215	15.143	0.698	0.139	19.562	–
		LC	56.572	24.625	8.902	14.543	0.686	1.136	29.648	0.097
		NC	88.741	49.010	16.077	27.297	5.281	1.097	36.350	0.176
Lotka-Volterra	PA	LC	41.140	19.071	7.358	10.775	0.591	0.592	20.843	0.074
		NC	46.656	25.837	9.276	12.451	3.918	0.519	19.431	0.101
Moore rot	PA	LC	1.645	1.076	0.447	0.455	0.097	0.097	0.391	0.012
		NC	2.640	2.057	0.715	0.655	0.660	0.087	0.381	0.016
Roessler	ID	FC	81.587	32.471	14.329	16.860	0.825	0.125	47.984	–
		LC	99.995	34.082	15.092	17.438	0.835	1.118	63.713	0.054
		NC	142.468	56.411	25.324	24.088	6.603	1.093	82.982	0.121
Vanderpol	NO	FC	79.781	78.919	37.866	40.271	0.243	–	–	–
		NC	89.488	88.641	43.358	43.047	1.648	–	–	–
Linear	ID	FC	0.088	0.072	0.032	0.033	0.006	0.000	0.008	–
		LC	0.125	0.082	0.035	0.036	0.006	0.006	0.031	0.001
		NC	0.198	0.156	0.054	0.051	0.050	0.006	0.027	0.001
Lin-dep	ID	FC	0.364	0.227	0.120	0.088	0.007	0.035	0.087	–
		LC	0.331	0.237	0.117	0.096	0.007	0.009	0.075	0.002
		NC	0.343	0.248	0.096	0.097	0.052	0.008	0.074	0.001
Sqr-deg	ID	FC	3.216	2.549	1.169	1.265	0.081	0.008	0.576	–
		LC	3.960	2.575	1.174	1.263	0.081	0.090	1.215	0.009
		NC	5.554	4.169	1.672	1.659	0.812	0.081	1.186	0.013
Pairwise	ID	FC	5.511	2.130	0.882	1.174	0.047	0.008	3.320	–
		LC	6.091	2.125	0.862	1.151	0.047	0.054	3.854	0.003
		NC	6.911	2.854	1.143	1.366	0.326	0.051	3.913	0.007

Table 7.3: Composition experiments for continuous and artificial systems.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Total Int : total integration, Poly : polynomial, Val : validating, Ref : refinement, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2.

Example	Proc	Comp	Time spent as % of its counterpart in NC							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
AND-Gate	ID	FC	122.8%	129.9%	193.3%	99.8%	54.3%	55.4%	100.3%	–
		LC	125.2%	132.3%	195.0%	102.5%	56.4%	107.3%	103.5%	345.8%
AND-OR Gate	ID	FC	46.4%	36.6%	76.9%	51.5%	7.6%	269.1%	61.2%	–
		LC	59.3%	41.2%	79.3%	59.0%	7.7%	143.5%	108.9%	178.9%
Brusselator	ID	FC	57.4%	55.2%	62.5%	69.7%	14.1%	17.1%	66.0%	–
		LC	70.4%	58.0%	64.4%	71.3%	14.3%	113.0%	102.5%	62.0%
Buckling col	NO	FC	73.8%	73.5%	82.7%	69.9%	14.6%	–	–	–
Jet engine	PA	LC	87.7%	72.1%	76.9%	85.8%	14.2%	119.6%	101.3%	79.2%
Lorentz	ID	FC	52.2%	52.4%	57.3%	55.5%	13.2%	12.7%	53.8%	–
		LC	63.7%	50.2%	55.4%	53.3%	13.0%	103.5%	81.6%	55.0%
Lotka-Volterra	PA	LC	88.2%	73.8%	79.3%	86.5%	15.1%	114.1%	107.3%	72.8%
Moore rot	PA	LC	62.3%	52.3%	62.6%	69.5%	14.7%	112.0%	102.5%	72.4%
Roessler	ID	FC	57.3%	57.6%	56.6%	70.0%	12.5%	11.5%	57.8%	–
		LC	70.2%	60.4%	59.6%	72.4%	12.6%	102.2%	76.8%	44.4%
Vanderpol	NO	FC	89.2%	89.0%	87.3%	93.5%	14.7%	–	–	–
Linear	ID	FC	44.3%	46.3%	59.1%	64.6%	11.0%	7.1%	30.6%	–
		LC	63.3%	52.1%	64.4%	70.7%	12.4%	116.1%	112.4%	81.6%
Lin-dep	ID	FC	106.1%	91.6%	124.4%	90.6%	14.3%	433.4%	117.6%	–
		LC	96.4%	95.6%	121.5%	99.3%	13.8%	112.2%	101.4%	125.2%
Sqr-deg	ID	FC	57.9%	61.1%	69.9%	76.3%	9.9%	9.7%	48.5%	–
		LC	71.3%	61.8%	70.2%	76.1%	10.0%	111.3%	102.5%	75.2%
Pairwise	ID	FC	79.7%	74.6%	77.1%	85.9%	14.3%	16.3%	84.8%	–
		LC	88.1%	74.5%	75.4%	84.3%	14.4%	106.7%	98.5%	47.4%

Table 7.4: Performance of composition compared to non-composition for continuous and artificial systems.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2, Poly : polynomial, Val : validating, Ref : refinement.

Example	Proc	Comp	Times							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Bouncing ball	ID	FC	0.337	0.236	0.105	0.098	0.011	0.004	0.064	–
		LC	0.489	0.250	0.106	0.104	0.011	0.021	0.191	0.008
		NC	0.664	0.422	0.176	0.153	0.086	0.021	0.182	0.009
Cruise control	ID	FC	4.861	3.059	1.439	1.193	0.098	0.084	1.259	–
		LC	7.035	3.201	1.381	1.262	0.098	0.276	3.180	0.124
		NC	9.013	4.993	2.269	1.788	0.845	0.266	3.235	0.118
Glycemic 1	ID	FC	45.964	35.008	9.352	17.662	4.340	0.714	6.997	–
		LC	82.668	37.656	9.574	19.154	4.317	3.114	38.678	1.242
		NC	214.930	175.853	16.898	88.564	69.556	2.146	33.073	0.869
Glycemic 2	ID	FC	57.801	44.568	8.737	26.470	7.067	0.274	9.978	–
		LC	101.538	46.984	8.884	27.422	7.278	5.670	45.677	0.847
		NC	207.055	161.565	17.639	89.216	53.781	2.417	38.658	0.930
Filtered osc 4	ID	FC	11.626	10.623	6.849	3.080	0.365	0.051	0.374	–
		LC	13.998	10.956	6.785	3.325	0.362	0.491	2.143	0.059
		NC	28.160	25.218	10.677	5.321	9.049	0.470	1.913	0.032
Filtered osc 8	ID	FC	27.808	25.948	17.207	7.161	0.584	0.124	0.900	–
		LC	35.503	27.662	17.413	7.925	0.580	1.281	5.648	0.082
		NC	84.753	76.374	31.953	15.544	28.440	1.254	5.827	0.061
Filtered osc 16	ID	FC	65.882	61.430	41.236	16.435	1.041	0.553	2.325	–
		LC	90.053	62.876	40.949	17.827	1.025	3.413	21.298	0.162
		NC	292.519	264.607	108.351	46.953	107.972	3.442	20.983	0.133
Filtered osc 32	ID	FC	141.936	126.936	86.851	33.849	2.281	3.480	6.177	–
		LC	273.422	134.788	88.096	37.778	2.237	10.721	120.549	0.432
		NC	1497.620	1366.910	582.293	184.488	596.053	11.322	108.219	0.389

Table 7.5: Composition experiments for the initial modes of hybrid systems (part 1).

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Total Int : total integration, Poly : polynomial, Val : validating, Ref : refinement, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2.

Example	Proc	Comp	Times							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Neuron 1	ID	FC	186.730	96.579	36.757	52.591	5.028	0.792	84.711	–
		LC	219.419	95.615	36.566	50.996	4.947	4.318	114.785	0.441
		NC	282.199	154.597	52.747	66.717	33.868	3.793	115.535	0.784
Neuron 2	ID	FC	17.845	9.874	3.753	5.364	0.466	0.073	7.468	–
		LC	21.974	10.058	3.868	5.470	0.474	0.439	10.988	0.053
		NC	27.401	15.637	5.329	7.082	3.108	0.391	10.609	0.076
Non-holonomic	ID	FC	3.967	2.889	1.407	1.095	0.121	0.048	0.642	–
		LC	5.866	2.977	1.313	1.133	0.114	0.202	2.369	0.161
		NC	8.066	5.090	2.375	1.663	0.950	0.153	2.402	0.125
Rod reactor	ID	FC	3.196	2.486	1.090	1.136	0.123	0.022	0.367	–
		LC	4.742	2.693	1.097	1.150	0.122	0.285	1.516	0.043
		NC	7.539	5.342	2.151	1.924	1.187	0.247	1.586	0.062
Switching	ID	FC	20.639	20.454	16.031	4.341	0.050	0.000	0.034	–
		LC	21.307	20.864	16.226	4.476	0.050	0.162	0.145	0.003
		NC	29.690	29.213	23.326	5.532	0.290	0.148	0.148	0.004
Two tanks	ID	FC	8.158	7.602	5.269	2.091	0.057	0.015	0.219	–
		LC	9.253	7.908	5.389	2.240	0.057	0.315	0.754	0.040
		NC	10.632	9.392	6.320	2.637	0.352	0.277	0.613	0.024
Three vehicle	NO	FC	1625.950	1607.000	1126.150	452.524	16.848	–	–	–
		NC	2957.220	2931.220	2091.200	671.161	153.255	–	–	–

Table 7.6: Composition experiments for the initial modes of hybrid systems (part 2).

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Total Int : total integration, Poly : polynomial, Val : validating, Ref : refinement, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2.

Example	Proc	Comp	Time spent as % of its counterpart in NC							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Bouncing ball	ID	FC	50.8%	56.0%	60.0%	64.1%	12.2%	20.6%	35.3%	–
		LC	73.6%	59.2%	60.1%	67.7%	12.5%	103.6%	104.8%	92.4%
Cruise control	ID	FC	53.9%	61.3%	63.4%	66.7%	11.6%	31.4%	38.9%	–
		LC	78.1%	64.1%	60.9%	70.5%	11.6%	103.7%	98.3%	104.8%
Glycemic 1	ID	FC	21.4%	19.9%	55.3%	19.9%	6.2%	33.3%	21.2%	–
		LC	38.5%	21.4%	56.7%	21.6%	6.2%	145.1%	116.9%	143.0%
Glycemic 2	ID	FC	27.9%	27.6%	49.5%	29.7%	13.1%	11.3%	25.8%	–
		LC	49.0%	29.1%	50.4%	30.7%	13.5%	234.6%	118.2%	91.1%
Filtered osc 4	ID	FC	41.3%	42.1%	64.1%	57.9%	4.0%	11.0%	19.5%	–
		LC	49.7%	43.4%	63.5%	62.5%	4.0%	104.5%	112.0%	186.4%
Filtered osc 8	ID	FC	32.8%	34.0%	53.9%	46.1%	2.1%	9.9%	15.4%	–
		LC	41.9%	36.2%	54.5%	51.0%	2.0%	102.1%	96.9%	134.7%
Filtered osc 16	ID	FC	22.5%	23.2%	38.1%	35.0%	1.0%	16.1%	11.1%	–
		LC	30.8%	23.8%	37.8%	38.0%	0.9%	99.2%	101.5%	122.3%
Filtered osc 32	ID	FC	9.5%	9.3%	14.9%	18.3%	0.4%	30.7%	5.7%	–
		LC	18.3%	9.9%	15.1%	20.5%	0.4%	94.7%	111.4%	111.1%
Neuron 1	ID	FC	66.2%	62.5%	69.7%	78.8%	14.8%	20.9%	73.3%	–
		LC	77.8%	61.8%	69.3%	76.4%	14.6%	113.9%	99.4%	56.3%
Neuron 2	ID	FC	65.1%	63.1%	70.4%	75.7%	15.0%	18.6%	70.4%	–
		LC	80.2%	64.3%	72.6%	77.2%	15.3%	112.4%	103.6%	70.4%
Non-holonomic	ID	FC	49.2%	56.8%	59.3%	65.9%	12.7%	31.1%	26.7%	–
		LC	72.7%	58.5%	55.3%	68.2%	12.0%	131.9%	98.6%	128.8%
Rod reactor	ID	FC	42.4%	46.5%	50.7%	59.0%	10.3%	8.8%	23.1%	–
		LC	62.9%	50.4%	51.0%	59.7%	10.3%	115.5%	95.6%	69.2%
Switching	ID	FC	69.5%	70.0%	68.7%	78.5%	17.3%	0.2%	23.1%	–
		LC	71.8%	71.4%	69.6%	80.9%	17.1%	109.5%	97.7%	71.5%
Two tanks	ID	FC	76.7%	80.9%	83.4%	79.3%	16.1%	5.5%	35.8%	–
		LC	87.0%	84.2%	85.3%	84.9%	16.1%	113.6%	123.1%	170.3%
Three vehicle	NO	FC	55.0%	54.8%	53.9%	67.4%	11.0%	–	–	–

Table 7.7: Performance of composition compared to non-composition for the initial modes of hybrid systems.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2, Poly : polynomial, Val : validating, Ref : refinement.

Example	Proc	Comp	Times							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Bouncing ball	ID	FC	0.038	0.025	0.012	0.010	0.001	0.001	0.008	–
		LC	0.043	0.026	0.011	0.011	0.001	0.002	0.011	0.001
		NC	0.041	0.023	0.011	0.009	0.002	0.002	0.010	0.001
Cruise control	ID	FC	0.499	0.296	0.143	0.118	0.012	0.009	0.143	–
		LC	0.558	0.309	0.137	0.128	0.011	0.025	0.175	0.014
		NC	0.502	0.256	0.128	0.101	0.016	0.028	0.160	0.010
Glycemic 1	ID	FC	4.470	3.287	0.923	1.708	0.445	0.072	0.722	–
		LC	5.494	3.885	1.031	2.037	0.470	0.216	1.056	0.127
		NC	5.532	4.052	0.779	2.302	0.911	0.181	0.902	0.067
Glycemic 2	ID	FC	5.598	4.200	0.848	2.538	0.712	0.029	1.011	–
		LC	6.079	4.338	0.841	2.554	0.697	0.210	1.223	0.088
		NC	5.784	4.054	0.823	2.474	0.695	0.196	1.114	0.065
Filtered osc 4	ID	FC	1.124	1.050	0.676	0.303	0.036	0.003	0.036	–
		LC	1.167	1.062	0.661	0.323	0.036	0.032	0.045	0.003
		NC	1.010	0.896	0.482	0.301	0.106	0.029	0.043	0.002
Filtered osc 8	ID	FC	2.751	2.564	1.717	0.711	0.059	0.012	0.088	–
		LC	2.902	2.670	1.699	0.774	0.059	0.070	0.104	0.006
		NC	2.358	2.099	1.075	0.716	0.293	0.062	0.105	0.003
Filtered osc 16	ID	FC	6.368	5.904	4.000	1.584	0.103	0.054	0.230	–
		LC	6.675	6.103	3.990	1.760	0.104	0.162	0.274	0.011
		NC	5.973	5.382	2.587	1.685	1.066	0.143	0.263	0.009
Filtered osc 32	ID	FC	14.182	12.685	8.705	3.387	0.226	0.332	0.622	–
		LC	15.064	13.486	8.855	3.768	0.226	0.387	0.875	0.027
		NC	18.369	16.819	6.741	4.074	5.904	0.356	0.766	0.019
Non-holonomic	ID	FC	0.400	0.290	0.137	0.107	0.014	0.005	0.070	–
		LC	0.467	0.312	0.140	0.119	0.013	0.017	0.097	0.018
		NC	0.397	0.251	0.141	0.084	0.017	0.015	0.086	0.011
Rod reactor	ID	FC	0.332	0.254	0.113	0.114	0.013	0.002	0.040	–
		LC	0.367	0.253	0.110	0.114	0.013	0.022	0.065	0.005
		NC	0.368	0.254	0.115	0.111	0.021	0.021	0.058	0.005
Two tanks	ID	FC	0.801	0.742	0.517	0.204	0.006	0.001	0.023	–
		LC	0.840	0.756	0.516	0.218	0.006	0.025	0.035	0.003
		NC	0.706	0.623	0.415	0.196	0.006	0.023	0.028	0.002

Table 7.8: Composition experiments for the initial modes of naturally compositional hybrid systems.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Total Int : total integration, Poly : polynomial, Val : validating, Ref : refinement, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2.

Example	Proc	Comp	Time spent as % of its counterpart in NC							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Bouncing ball	ID	FC	92.7%	109.1%	104.8%	111.2%	67.7%	23.0%	73.4%	–
		LC	104.7%	112.7%	101.0%	120.6%	68.3%	95.9%	102.0%	118.1%
Cruise control	ID	FC	99.4%	115.5%	111.9%	116.6%	72.4%	32.5%	89.2%	–
		LC	111.1%	120.7%	107.5%	126.9%	70.3%	91.0%	109.7%	135.7%
Glycemic 1	ID	FC	80.8%	81.1%	118.5%	74.2%	48.8%	39.6%	80.1%	–
		LC	99.3%	95.9%	132.5%	88.5%	51.5%	119.0%	117.1%	188.7%
Glycemic 2	ID	FC	96.8%	103.6%	103.0%	102.6%	102.5%	14.6%	90.8%	–
		LC	105.1%	107.0%	102.1%	103.2%	100.3%	107.0%	109.8%	134.2%
Filtered osc 4	ID	FC	111.2%	117.2%	140.2%	100.9%	34.2%	11.4%	84.7%	–
		LC	115.5%	118.5%	137.1%	107.5%	33.9%	107.6%	103.9%	161.1%
Filtered osc 8	ID	FC	116.7%	122.2%	159.7%	99.3%	20.1%	18.8%	83.9%	–
		LC	123.1%	127.2%	158.1%	108.2%	20.2%	113.5%	99.1%	177.9%
Filtered osc 16	ID	FC	106.6%	109.7%	154.6%	94.0%	9.6%	37.9%	87.7%	–
		LC	111.8%	113.4%	154.2%	104.5%	9.8%	113.0%	104.3%	134.4%
Filtered osc 32	ID	FC	77.2%	75.4%	129.1%	83.1%	3.8%	93.2%	81.2%	–
		LC	82.0%	80.2%	131.4%	92.5%	3.8%	108.5%	114.2%	143.9%
Non-holonomic	ID	FC	100.8%	115.4%	97.1%	126.4%	79.1%	33.2%	81.6%	–
		LC	117.5%	124.0%	98.9%	141.2%	76.2%	118.7%	112.9%	166.8%
Rod reactor	ID	FC	90.1%	99.9%	97.8%	102.8%	63.1%	11.2%	68.1%	–
		LC	99.7%	99.3%	95.5%	102.7%	62.7%	104.4%	112.2%	105.2%
Two tanks	ID	FC	113.5%	119.2%	124.6%	104.0%	102.7%	6.4%	81.4%	–
		LC	119.0%	121.4%	124.3%	110.8%	99.8%	107.9%	123.3%	138.0%

Table 7.9: Performance of composition compared to non-composition for the initial modes of naturally compositional hybrid systems.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2, Poly : polynomial, Val : validating, Ref : refinement.

For those systems that can make use of the compositional approach without needing to include copies[3], we present the number of refinements in Figures 7.17 and 7.18. We plot how many numbers of refinements[4] each component required at any time step.

When presenting the data we call both the compositional naive algorithm (i.e. method with no processing) and the preconditioning algorithm using compositional preconditioning[5] *fully compositional*. We call the preconditioning method with compositional integration and non-compositional preconditioning *left model compositional*.

We note that our current setup treats the non-compositional approach as having everything in a single component. This has the effect of adding some insignificant overhead in the non-compositional approach (observed in mapping 1 and mapping 2).

We also note that when we use fully compositional algorithms, we do not need to apply any mappings after preconditioning (mapping 2).

7.4.1 Analysis of Composition Experiments

7.4.1.1 Continuous and artificial systems

We see some expected things in Tables 7.3 and 7.4:

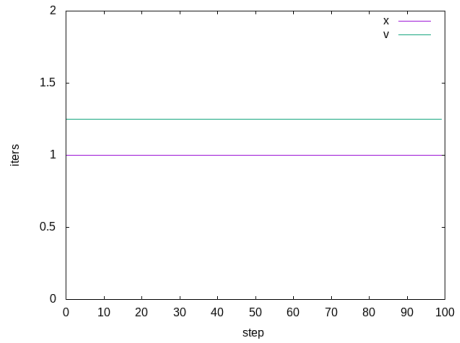
- similar performances in integration phases of the left compositional methods and the fully compositional method,
- similar performances in preconditioning phases of the left compositional methods and non-compositional methods,
- the overhead from the composition (mapping 1 and mapping 2) is insignificant (in general).

Inspecting the data for Brusselator, Buckling column, Jet engine, Lorentz, Lotka-Volterra, Moore's rotational, Roessler attractor, Van der Pol oscillator, Linear compositional, Squared degradation and Pairwise dependent we can see that they get similar performance gains. This is a mixture of two kinds of systems: continuous and artificial. The continuous systems are not compositional in their nature and are made compositional by having 10 copies of these systems in a larger system. This makes

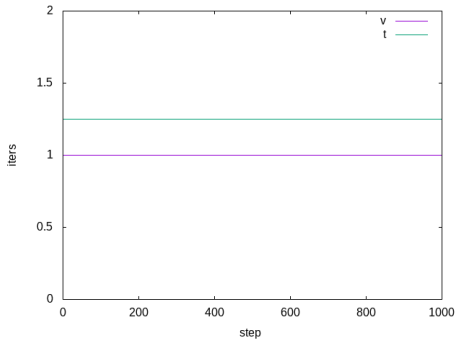
[3] We do not plot the number of refinements for other systems, since all of the components would have the exact same number of refinements.

[4] To make equivalent data points more readable, we shift components based on their index w.r.t. to the y axis. All the y axis values should be considered floored.

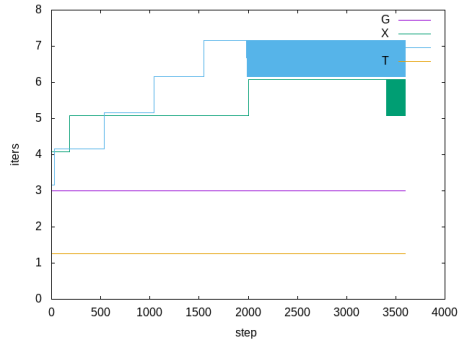
[5] Compositional preconditioning also requires compositional integration.



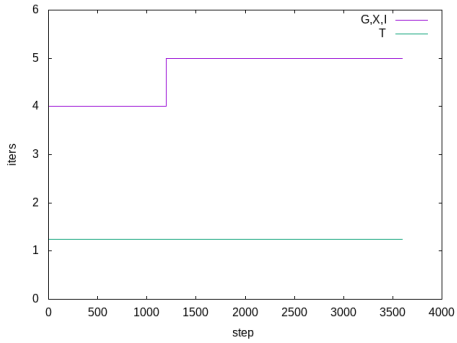
(a) Bouncing ball



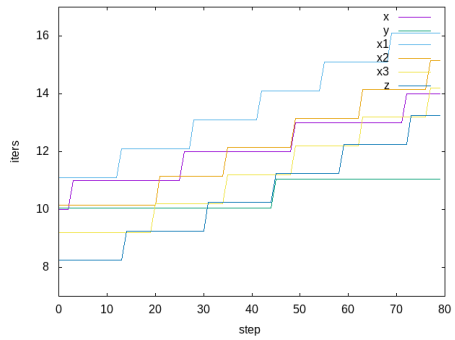
(b) Cruise control



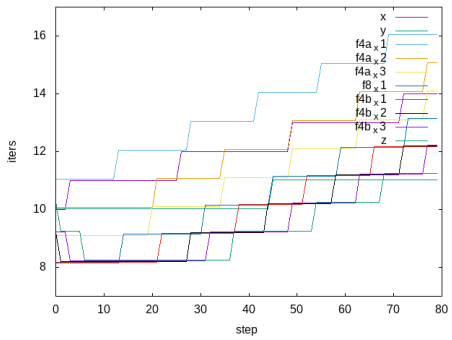
(c) Glycemic 1



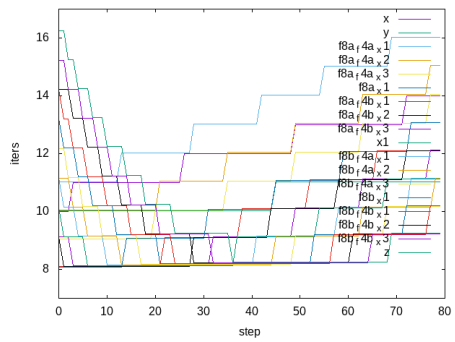
(d) Glycemic 2



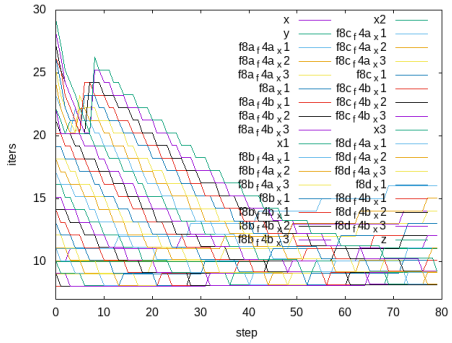
(e) Filtered osc 4



(f) Filtered osc 8

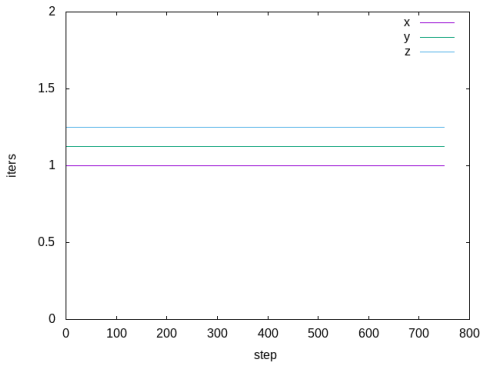


(g) Filtered osc 16

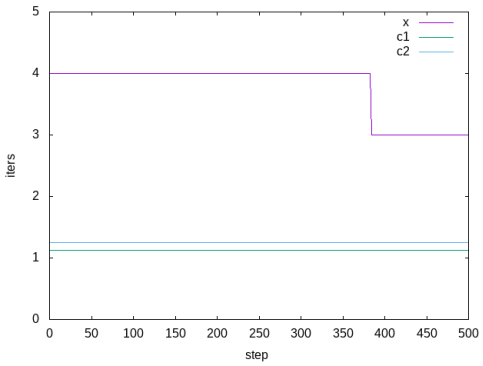


(h) Filtered osc 32

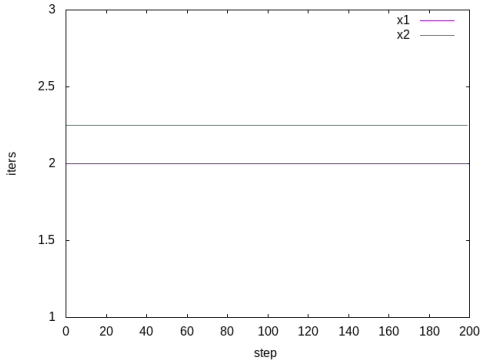
Figure 7.17: Number of refinements per component.



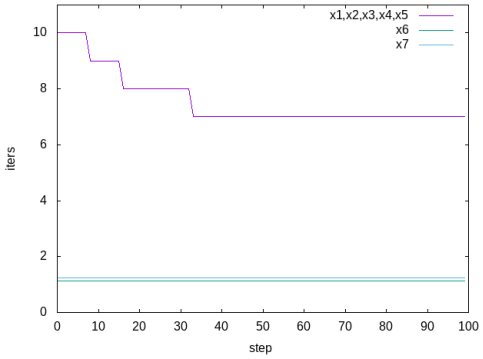
(a) Non-holonomic



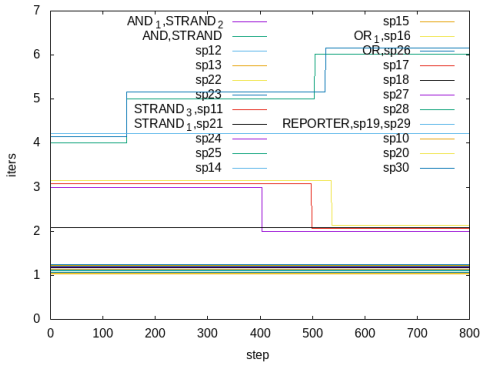
(b) Rod reactor



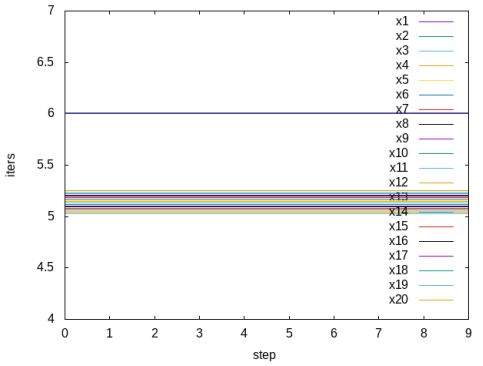
(c) Two tanks



(d) AND-Gate



(e) AND-OR Gate



(f)

Figure 7.18: Number of refinements per component.

them very similar to artificial systems. For all of these systems, we can say that the dynamics of the system do not play a big factor in the differences in compositional or non-compositional methods, only the data structures. The performance gains we see here are that total time with composition is about 70% of the total time without composition, 60% for integration and around 60% for compositional preconditioning (visible in the preconditioning column for left model compositional identity preconditioning). The most drastic change here is that refining the remainder term in integration takes only about 14% with composition.

Turning to the de-stiffened AND-Gate, we observe that the time spent computing the polynomial during Picard iteration is significantly larger with the composition (about 2 times). This is likely due to the fact that in two components we are using a precomputed higher-order polynomial from its dependency. Operating on this higher-order polynomial involves more operations compared to lower-order polynomials in the non-compositional method[6].

Another reason for performance loss in AND-Gate might be that this system is not engineered to be compositional and it also reflects on the structure. The system consists of 7 variables, 5 of which are in a single component and 2 are in separate components. Majority of the dynamics of the systems is concentrated in the 5-variable components, the other 2 components are very simple in their nature. We can somewhat observe it from Figure 7.18 (d), where we can see that the bigger component requires between 7-10 refinements while the other two components require just one refinement. This means that the sizes of data structures are not optimally distributed between components.

The linearly dependent system is engineered in some ways to be unsuitable for the compositional approach. This is due to the fact that the components have cascading dependency and there is a build-up of the number of parameters in components as they appear later in the topological sort. This reduces performance gains in the integration phase. In fully compositional preconditioning the effects of more efficient preconditioning are reduced by the larger amount of flowpipes that need to mapped (mapping 1).

Finally, we can see that composition yields significant performance gains on the

[6]Our implementation always uses k^{th} order Picard iterations for any influencer outside of the current component. This has the effect that we need to store and map less Taylor models, but we introduce unnecessary terms when computing lower-order Picard iterations. We could get rid of this peculiarity if we were to store all of the lower-order Picard iterations and map them into appropriate variables when using them later. However, this would increase the memory requirements and time spent in mapping.

AND-OR-Gate model and Linear compositional, with the fully compositional approach taking less than 50% of the time of the non-compositional method.

Let us first focus on AND-OR-Gate. This system can be partitioned into two independent parts which are particularly suitable for composition[7] on the integration phase. These independent systems share a key characteristic with the linearly dependent system in that there are a high number of components with many dependencies, which results in a performance hit for mapping in compositional preconditioning. We also observe massive performance gains in the refinement part. This is likely due to the fact that the system includes 15 one-variable components (out of 22). With composition, the refinement part in these 15 components is affected by just one variable, meaning that if we can find a suitable remainder for this one variable, we are done with refinement part for this component all together (cf. (f) in Figure 7.18). In the non-compositional approach, whenever any of the variables needs still to be refined, we are computing new remainders for all of the variables in the system.

Linear dependent is most distinct by having much more expensive mapping of data structures (mapping 1), this is because the Taylor model computed in the earlier component need to mapped into appropriate dimensions to be used in later components. Since component influence here is completely linear, there needs to be $\frac{n(n-1)}{2}$ mappings for the n-dimensional system.

7.4.1.2 Initial modes of hybrid systems

The experiments involving initial modes of hybrid systems are similar to the ones involving continuous systems but offer bigger performance gains in some cases. To understand this let us describe these systems in more detail. These systems contain only one mode of a hybrid system. This one mode contains only one type of dynamic, the system, however, is designed to have other dynamics too. This leads to the case that when computing flowpipes for variables, some of them require more work (the ones that are more active in the initial mode). Which can be observed most drastically in the refinement part for the same reason as discussed above.

Like AND-Gate some of these systems are not designed from components but do include some variables that can be viewed as such. In these systems, our strategy of copying having 10 copies of the system to see it as a compositional system, results in two-fold composition (one from coping and one from the original structure). This

[7] Although these independent parts compute AND functions, they do not resemble the de-stiffened AND-Gate.

helps to explain somewhat larger gains we see in Tables 7.5 to 7.7. Other than that these experiments are similar to experiments about Brusselator, Buckling column, Jet engine, Lorentz, Lotka-Volterra, Moore's rotational, Roessler attractor, Van der Pol oscillator, Linear compositional, Squared degradation and Pairwise dependent in the previous section.

Let us turn our attention to the systems where we can apply the compositional approach without needing to add copies. The polynomial and validating parts in Tables 7.8 or 7.9 show that composition loses performance often in these cases. This is likely due to the same reason as to why it also happened with AND-Gate.

Most systems show performance gains in the refinement part. This is likely due to the same reason as we discussed in AND-OR Gate, i.e. that in addition to using smaller data structures, different components might require the different number of refinements (cf. Figures 7.17 and 7.18).

Systems Bouncing ball, Cruise control, Non-holonomic and Two tanks happen to refine every component the same number of times. This is also mirrored in Tables 7.8 or 7.9 where refinement part does not show significant performance gains.

Glucemic 1 shows moderate speed up on the refinement part. From Figure 7.17 (c) we can see that the number of refinements required per component are also moderately apart.

Glycemic 2 and Rod reactor are similar in the way that they have a single component with complex dynamics and other component(s) with very simple dynamics (the derivative is constant). The simple components require much fewer refinements compared to the complex one. However, the simple components also have very simple Taylor model which results in very cheap refinement part. We do not see big performance gains here since in both compositional and non-compositional methods, the refinement part is dominated by the complex component. Unlike AND-Gate, these systems do not suffer from much slower time polynomial step. In AND-Gate, the complex component was influencing the simple ones and as a result, unnecessary complex Taylor models were used during the Picard iteration. In here, the simple components are influencing the complex ones and as a result, the Taylor models that get used in the Picard iteration are also simple.

Filtered oscillators show significant performance gains that increase as the system gets larger. This is mirrored in Figure 7.17 (e)-(h) where we can see that as the systems get larger, the difference of refinements required per component also gets larger.

7.5 Further Performance Gains

We expected larger gains from compositional than what we observed in the previous section. We found two explanations for this.

The first reason is that the polynomials in CFlow* (and in Flow*) are represented using Horner forms. The idea of composition is to eliminate the need to allocate memory for variables and parameters that do not appear when considering specific variables. Horner forms also do that to a degree. When a polynomial is represented using a Horner form, the variables that do not appear there are cut off (we are using the same implementation for Horner forms as Flow*). For example, in the system with variable x_1, x_2 and x_3, the polynomial x_1^2 is represented as

$$0+x_1\Big(0+x_1\Big(1+x_1(0)+x_2(0)+x_3(0)\Big)+x_2\Big(0\Big)+x_3\Big(0\Big)\Big)+x_2\Big(0\Big)+x_3\Big(0\Big)$$

where you can see that non-existent terms have been cut off. This reduces the effect of composition since the variables that can be eliminated are less relevant in the polynomials represented with Horner forms.

We found the second reason using the profiling tool gprof [GKM82]. With that tool, we found that the majority of the time is spent on the arithmetic operations on the monomials. In our tool CFlow* (and in Flow* from where we inherited the implementation), the operations on monomials consist of two parts: operation on coefficient and operation on the powers of the variables. The coefficients are represented as the MPFR numbers and therefore the former reduces to operations on the MPFR numbers. It turned out that they are very expensive compared to the operations on the powers of the variables. Unfortunately, the compositional approach does not affect operation on coefficients at all.

We plot the time cost of different operations in Figure 7.19. We can see that in order to have the operation on coefficient not dominate the operation on the monomial we need more than 400 dimensions for multiplication and a lot more than that for addition.

We noted that the systems that we have experimented on are not sensitive to the precision of the MPFR numbers. The integration breaks down due to the over-estimation introduced that stems mainly from the dynamics of the system and not the rounding errors of the MPFR numbers. The systems do not require more precision than what is the precision of the double-precision floating-point numbers(53 bits). In other words, we could use plain double-precision numbers instead of the MPFR numbers as long as we use the proper rounding mode to guarantee the correctness of the interval arith-

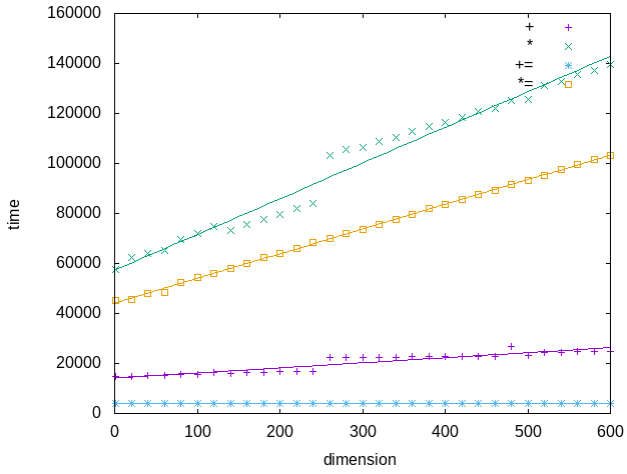


Figure 7.19: Time of 10000 monomial operations with MPFR.

metic. We have implemented this change. The results of the new implementation are in Tables 7.10, 7.11, 7.12, 7.13 and 7.14. From these tables we can see:

- an overall performance gain of around 5 times with the new implementation,
- the same overall trends that were there with MPFR implementation,
- slightly fewer performance gains from the compositional approach compared to the non-compositional approach.

The new implementation requires less computational power when multiplying coefficients, but it is still dominating in most cases (cf. Figures 7.20 and 7.21). This is because it needs to change the rounding mode of the processor often and changing the rounding mode is an expensive operation.

7.6 Shrink Wrapping

In order to evaluate shrink wrapping method, let use compare it to other processing methods.

More precisely, we will compute flows using

- identity preconditioning,
- parallelepiped preconditioning,
- QR preconditioning,

Example	Proc	Comp	Times							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
AND-Gate	ID	FC	9.224	7.852	3.931	3.832	0.007	0.022	1.271	–
		LC	9.444	8.060	3.912	4.054	0.007	0.047	1.304	0.003
		NC	7.110	5.720	1.925	3.774	0.012	0.044	1.290	0.002
AND-OR Gate	ID	FC	4.946	2.782	0.935	1.208	0.096	0.598	1.174	–
		LC	6.073	3.022	0.915	1.287	0.095	0.282	2.440	0.132
		NC	7.354	4.317	1.182	2.235	0.831	0.254	2.382	0.073
Brusselator	ID	FC	1.302	0.861	0.310	0.516	0.015	0.008	0.380	–
		LC	1.740	0.946	0.327	0.539	0.016	0.057	0.691	0.008
		NC	2.335	1.470	0.559	0.810	0.092	0.058	0.704	0.010
Buckling col	NO	FC	19.194	18.808	8.570	9.721	0.040	–	–	–
		NC	25.347	24.923	11.183	13.140	0.155	–	–	–
Jet engine	PA	LC	9.246	3.528	1.377	2.000	0.030	0.170	5.417	0.025
		NC	10.178	4.619	1.892	2.519	0.174	0.162	5.128	0.021
Lorentz	ID	FC	9.360	5.422	2.266	2.864	0.044	0.049	3.617	–
		LC	14.333	5.793	2.474	3.078	0.045	0.514	7.625	0.052
		NC	18.729	10.335	4.554	5.368	0.285	0.507	7.174	0.062
Lotka-Volterra	PA	LC	8.934	4.132	1.784	2.176	0.036	0.244	4.375	0.035
		NC	10.475	5.689	2.559	2.871	0.208	0.238	4.223	0.032
Moore rot	PA	LC	0.409	0.257	0.128	0.093	0.007	0.044	0.086	0.005
		NC	0.546	0.387	0.207	0.139	0.033	0.040	0.080	0.006
Roessler	ID	FC	17.904	8.087	3.926	3.933	0.047	0.044	9.487	–
		LC	25.656	8.324	3.986	4.006	0.048	0.518	16.459	0.023
		NC	31.877	14.219	7.253	6.461	0.367	0.490	16.531	0.039
Vanderpol	NO	FC	19.205	18.947	9.518	9.198	0.018	–	–	–
		NC	24.563	24.268	12.158	11.707	0.086	–	–	–
Linear	ID	FC	0.029	0.022	0.011	0.009	0.001	0.000	0.003	–
		LC	0.036	0.023	0.011	0.009	0.001	0.004	0.007	0.000
		NC	0.041	0.030	0.016	0.011	0.003	0.003	0.006	0.001
Lin-dep	ID	FC	0.097	0.060	0.036	0.018	0.001	0.018	0.013	–
		LC	0.092	0.068	0.039	0.021	0.001	0.005	0.016	0.001
		NC	0.077	0.053	0.029	0.021	0.002	0.004	0.016	0.001
Sqr-deg	ID	FC	0.662	0.532	0.270	0.239	0.007	0.004	0.098	–
		LC	0.837	0.543	0.272	0.238	0.007	0.039	0.233	0.004
		NC	1.147	0.847	0.446	0.352	0.042	0.038	0.220	0.005
Pairwise	ID	FC	1.162	0.505	0.232	0.259	0.003	0.003	0.638	–
		LC	1.504	0.539	0.239	0.270	0.003	0.025	0.923	0.002
		NC	1.608	0.679	0.323	0.333	0.018	0.023	0.871	0.002

Table 7.10: Composition experiments for continuous and artificial systems with intervals using doubles.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Total Int : total integration, Poly : polynomial, Val : validating, Ref : refinement, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2.

Example	Proc	Comp	Time spent as % of its counterpart in NC							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
AND-Gate	ID	FC	129.7%	137.3%	204.1%	101.5%	63.1%	48.9%	98.5%	–
		LC	132.8%	140.9%	203.2%	107.4%	59.0%	106.9%	101.1%	181.1%
AND-OR Gate	ID	FC	67.3%	64.4%	79.1%	54.0%	11.6%	235.5%	49.3%	–
		LC	82.6%	70.0%	77.4%	57.6%	11.4%	110.9%	102.4%	181.5%
Brusselator	ID	FC	55.8%	58.5%	55.4%	63.7%	16.6%	14.3%	53.9%	–
		LC	74.5%	64.4%	58.5%	66.5%	17.1%	97.7%	98.1%	86.8%
Buckling col	NO	FC	75.7%	75.5%	76.6%	74.0%	25.8%	–	–	–
Jet engine	PA	LC	90.8%	76.4%	72.8%	79.4%	17.2%	105.4%	105.6%	117.2%
Lorentz	ID	FC	50.0%	52.5%	49.8%	53.4%	15.4%	9.6%	50.4%	–
		LC	76.5%	56.1%	54.3%	57.3%	15.7%	101.2%	106.3%	84.1%
Lotka-Volterra	PA	LC	85.3%	72.6%	69.7%	75.8%	17.1%	102.4%	103.6%	107.5%
Moore rot	PA	LC	74.8%	66.4%	61.7%	66.9%	20.3%	110.4%	107.0%	86.6%
Roessler	ID	FC	56.2%	56.9%	54.1%	60.9%	12.8%	9.1%	57.4%	–
		LC	80.5%	58.5%	55.0%	62.0%	13.0%	105.8%	99.6%	58.1%
Vanderpol	NO	FC	78.2%	78.1%	78.3%	78.6%	21.3%	–	–	–
Linear	ID	FC	69.8%	74.2%	68.2%	82.8%	28.1%	11.1%	50.6%	–
		LC	87.2%	77.0%	68.2%	82.6%	26.5%	136.7%	124.2%	93.2%
Lin-dep	ID	FC	125.0%	113.3%	124.4%	86.6%	26.2%	459.1%	85.0%	–
		LC	119.0%	129.5%	134.4%	101.9%	28.7%	113.7%	100.5%	192.8%
Sqr-deg	ID	FC	57.7%	62.8%	60.5%	68.0%	16.0%	9.6%	44.6%	–
		LC	73.0%	64.0%	61.1%	67.8%	16.2%	104.3%	105.8%	84.5%
Pairwise	ID	FC	72.3%	74.5%	71.8%	78.0%	15.8%	13.9%	73.3%	–
		LC	93.6%	79.5%	73.9%	81.2%	16.8%	106.7%	106.0%	86.9%

Table 7.11: Performance of composition compared to non-composition for continuous and artificial systems with intervals using doubles.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2, Poly : polynomial, Val : validating, Ref : refinement.

Example	Proc	Comp	Times							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Bouncing ball	ID	FC	0.129	0.087	0.039	0.030	0.003	0.002	0.022	–
		LC	0.177	0.093	0.039	0.032	0.003	0.012	0.059	0.006
		NC	0.180	0.096	0.058	0.030	0.005	0.012	0.054	0.005
Cruise control	ID	FC	1.445	0.929	0.405	0.313	0.037	0.042	0.261	–
		LC	1.979	1.035	0.440	0.330	0.036	0.126	0.674	0.066
		NC	1.926	0.980	0.604	0.306	0.046	0.118	0.639	0.047
Glycemic 1	ID	FC	12.271	8.733	2.639	3.834	0.469	0.327	1.733	–
		LC	20.839	9.674	2.817	4.151	0.489	1.288	8.607	0.618
		NC	29.411	18.588	4.858	9.893	3.534	1.160	8.157	0.429
Glycemic 2	ID	FC	11.577	8.255	2.333	4.366	0.517	0.118	2.011	–
		LC	21.319	8.426	2.286	4.193	0.468	1.323	10.437	0.415
		NC	30.902	18.865	5.202	10.580	2.746	1.311	9.069	0.426
Filtered osc 4	ID	FC	3.060	2.775	2.001	0.618	0.025	0.017	0.072	–
		LC	3.946	2.852	1.977	0.658	0.026	0.268	0.693	0.020
		NC	7.472	6.463	4.276	1.611	0.509	0.261	0.529	0.016
Filtered osc 8	ID	FC	7.999	7.533	5.497	1.574	0.042	0.060	0.150	–
		LC	10.854	7.837	5.369	1.636	0.047	0.734	1.926	0.042
		NC	23.696	20.476	13.423	5.343	1.511	0.716	1.942	0.034
Filtered osc 16	ID	FC	20.169	18.998	13.869	3.819	0.082	0.315	0.375	–
		LC	31.611	18.398	13.017	3.867	0.083	2.251	9.695	0.092
		NC	96.766	82.963	53.565	20.297	8.399	2.164	9.775	0.083
Filtered osc 32	ID	FC	43.740	38.260	28.342	7.802	0.172	2.415	1.000	–
		LC	116.167	40.705	28.726	8.457	0.181	7.670	63.657	0.258
		NC	494.398	414.465	278.387	89.281	44.489	7.883	65.419	0.268

Table 7.12: Composition experiments for the initial modes of hybrid systems with intervals using doubles (part 1).

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Total Int : total integration, Poly : polynomial, Val : validating, Ref : refinement, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2.

Example	Proc	Comp	Times							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Neuron 1	ID	FC	38.841	21.308	9.486	10.492	0.344	0.302	15.704	–
		LC	49.773	21.104	9.213	10.201	0.349	1.764	25.444	0.223
		NC	61.803	30.682	14.344	14.284	1.686	1.781	25.953	0.248
Neuron 2	ID	FC	3.689	2.162	0.944	1.057	0.031	0.028	1.366	–
		LC	4.901	2.179	0.971	1.071	0.032	0.186	2.389	0.026
		NC	5.844	3.111	1.463	1.461	0.153	0.183	2.291	0.024
Non-holonomic	ID	FC	1.298	0.888	0.395	0.306	0.040	0.025	0.195	–
		LC	1.875	0.982	0.398	0.327	0.043	0.082	0.673	0.079
		NC	2.004	1.086	0.696	0.309	0.053	0.082	0.668	0.061
Rod reactor	ID	FC	1.032	0.731	0.333	0.283	0.028	0.013	0.141	–
		LC	1.631	0.816	0.326	0.282	0.029	0.141	0.575	0.031
		NC	2.015	1.154	0.651	0.399	0.066	0.141	0.572	0.041
Switching	ID	FC	5.139	5.109	4.151	0.948	0.002	0.000	0.005	–
		LC	5.162	5.021	4.036	0.949	0.002	0.072	0.038	0.001
		NC	10.189	10.027	8.201	1.783	0.018	0.075	0.038	0.002
Two tanks	ID	FC	2.318	2.160	1.629	0.442	0.009	0.007	0.048	–
		LC	2.455	2.048	1.518	0.430	0.009	0.144	0.199	0.014
		NC	2.714	2.348	1.750	0.555	0.018	0.128	0.120	0.010
Three vehicle	NO	FC	516.059	510.379	373.797	131.493	0.512	–	–	–
		NC	1261.330	1250.140	947.543	289.583	5.690	–	–	–

Table 7.13: Composition experiments for the initial modes of hybrid systems with intervals using doubles (part 2).

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Total Int : total integration, Poly : polynomial, Val : validating, Ref : refinement, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2.

Example	Proc	Comp	Time spent as % of its counterpart in NC							
			Total	Integration				Processing		
				Total Int	Poly	Val	Ref	Map 1	Precond	Map 2
Bouncing ball	ID	FC	71.7%	91.4%	67.4%	100.3%	67.9%	19.1%	39.7%	–
		LC	98.4%	96.8%	67.2%	106.2%	70.1%	101.7%	108.1%	125.5%
Cruise control	ID	FC	75.0%	94.7%	67.0%	102.3%	81.5%	35.4%	40.9%	–
		LC	102.8%	105.6%	72.9%	107.7%	79.4%	107.1%	105.6%	142.3%
Glycemic 1	ID	FC	41.7%	47.0%	54.3%	38.8%	13.3%	28.2%	21.2%	–
		LC	70.9%	52.0%	58.0%	42.0%	13.8%	111.0%	105.5%	144.0%
Glycemic 2	ID	FC	37.5%	43.8%	44.9%	41.3%	18.8%	9.0%	22.2%	–
		LC	69.0%	44.7%	44.0%	39.6%	17.0%	100.9%	115.1%	97.4%
Filtered osc 4	ID	FC	40.9%	42.9%	46.8%	38.4%	4.9%	6.4%	13.5%	–
		LC	52.8%	44.1%	46.2%	40.8%	5.1%	102.7%	131.1%	121.6%
Filtered osc 8	ID	FC	33.8%	36.8%	41.0%	29.5%	2.8%	8.3%	7.7%	–
		LC	45.8%	38.3%	40.0%	30.6%	3.1%	102.5%	99.2%	125.0%
Filtered osc 16	ID	FC	20.8%	22.9%	25.9%	18.8%	1.0%	14.6%	3.8%	–
		LC	32.7%	22.2%	24.3%	19.1%	1.0%	104.1%	99.2%	110.6%
Filtered osc 32	ID	FC	8.8%	9.2%	10.2%	8.7%	0.4%	30.6%	1.5%	–
		LC	23.5%	9.8%	10.3%	9.5%	0.4%	97.3%	97.3%	96.3%
Neuron 1	ID	FC	62.8%	69.4%	66.1%	73.5%	20.4%	17.0%	60.5%	–
		LC	80.5%	68.8%	64.2%	71.4%	20.7%	99.1%	98.0%	90.0%
Neuron 2	ID	FC	63.1%	69.5%	64.5%	72.4%	20.4%	15.0%	59.6%	–
		LC	83.9%	70.0%	66.4%	73.3%	20.7%	101.4%	104.3%	105.1%
Non-holonomic	ID	FC	64.8%	81.8%	56.7%	99.3%	75.1%	30.2%	29.2%	–
		LC	93.6%	90.4%	57.1%	105.9%	81.0%	100.5%	100.8%	129.0%
Rod reactor	ID	FC	51.2%	63.3%	51.1%	70.9%	42.7%	9.5%	24.6%	–
		LC	80.9%	70.7%	50.0%	70.7%	44.6%	100.4%	100.5%	75.5%
Switching	ID	FC	50.4%	51.0%	50.6%	53.2%	9.4%	0.2%	13.5%	–
		LC	50.7%	50.1%	49.2%	53.2%	9.9%	95.7%	99.5%	69.1%
Two tanks	ID	FC	85.4%	92.0%	93.1%	79.7%	51.5%	5.6%	40.0%	–
		LC	90.5%	87.2%	86.7%	77.6%	51.3%	113.1%	165.7%	139.9%
Three vehicle	NO	FC	40.9%	40.8%	39.4%	45.4%	9.0%	–	–	–

Table 7.14: Performance of composition compared to non-composition for the initial modes of hybrid systems with intervals using doubles.

Proc : processing method (identity, parallelepiped, none), Comp : composition type, FC : fully compositional, CL : left model compositional, NC : non-compositional, Map 1 : mapping 1, Precond : preconditioning without mapping, Map 2 : mapping 2, Poly : polynomial, Val : validating, Ref : refinement.

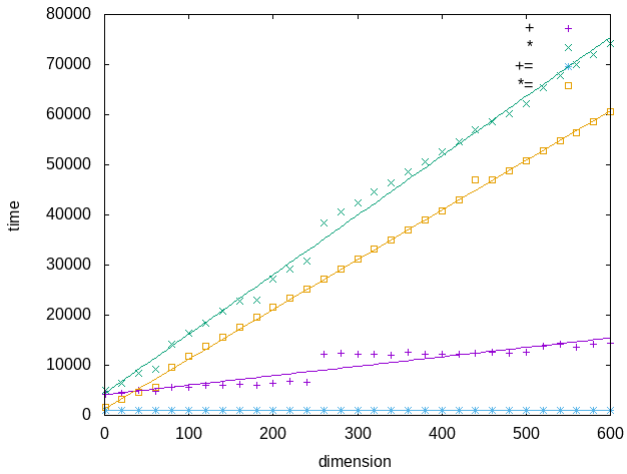


Figure 7.20: Time of 10000 monomial operations with doubles.

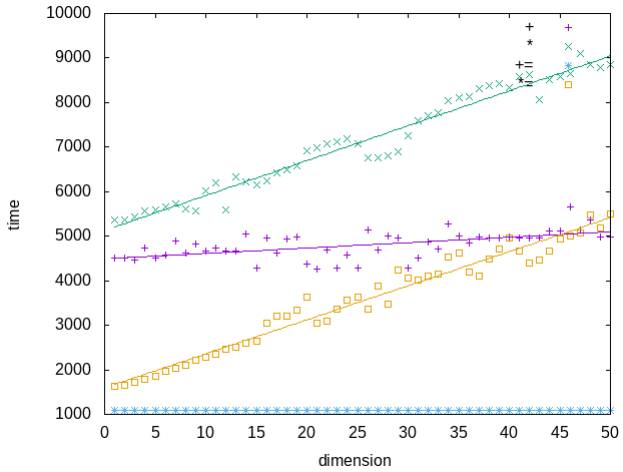


Figure 7.21: Time of 10000 monomial operations (dimensions 1-50) with doubles.

- no processing,
- shrink wrapping at every step,
- shrink wrapping after every 2 steps,
- shrink wrapping after every 5 steps,
- shrink wrapping after every 10 steps.

We present data for this in Table 7.15. We mark the best method as we described in Section 7.3. To conserve space, we do not show the widths of the intervals in this table (these can be found in Section B.2).

We can see from this table that in general shrink wrapping breaks down a lot earlier than most alternatives. We suspect that this is due to it not being very suitable for non-linear systems. We note that systems Linear compositional, Linear dependent, Cruise control and Three vehicle platoon where it manages to integrate as far as the other methods are linear and very simple.

The main reason why shrink wrapping breaks down is that it introduces a lot of overestimation when used on such systems. We try to explain applying shrink wrapping on non-linear systems with an informal discussion of applying it on the ODE system corresponding to a simple chemical reaction network.

A common dynamic in a chemical reaction network is to have two chemicals react with each other. Suppose we have a reaction r which combines two chemicals X_1 and X_2

$$r: \quad X + Y \xrightarrow{k} Z.$$

Let us use variables x and y to model the concentration of X and Y respectively. The reaction r will introduce a term kxy in both of the derivatives of these variables. Let us ignore everything else about this hypothetical system and focus on what happens with this effect. In other words, let us assume that we have a system with ODEs

$$\begin{aligned} \frac{dx}{dt} &= -kxy \\ \frac{dy}{dt} &= -kxy \end{aligned}$$

Suppose we have an interval box for the initial conditions for this system, then the flow of this system will evolve from the rectangle shape into the crescent shape (shown in Figure 7.22). The reason for this evolution is that the reaction r consumes

Example	Preconditioning method							
	Id	Pa	QR	Non	Sw1	Sw2	Sw5	Sw10
AND-Gate	**1000**	0	1000	570	10	40	50	100
AND-OR Gate	**40**	14.55	31.75	40	0.6	0.8	1.25	1.85
Brusselator	6	2.4	**15**	1.89	1.38	1.38	1.35	1.5
Buckling col	7.15	4.82	**10**	10	0.9	0.9	0.9	0.9
Jet engine	2.91	**10.02**	8.73	1.65	4.35	4.32	4.2	2.7
Lorentz	1.725	0.627	**5.397**	0.912	0.579	0.582	0.585	0.6
Lotka-Volterra	3.3	**10**	6.5	2.14	5.34	5.36	5.4	5.4
Moore rot	10	**10**	**10**	10	10	10	10	10
Roessler	6	0	**6**	6	0.02	0.04	0.04	0.2
Vanderpol	7	2.86	**7**	7	0.32	0.32	0.4	0.4
Linear	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Lin-dep	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Sqr-deg	**100**	**100**	**100**	100	18	18	20	20
Pairwise	60	45	**60**	60	24	24	30	30
Bouncing ball	**10**	0	**10**	**10**	0.1	0.2	1	2
Cruise control	**100.1**	**100.1**	**100.1**	**100.1**	**100.1**	**100.1**	**100.1**	**100.1**
Glycemic 1	**360**	0	360	2.9	0.1	0.2	1.8	2.4
Glycemic 2	**360**	0	360	2.9	0.1	0.2	1.8	2.4
Filtered osc 4	**4**	0	**4**	4	0.05	2.8	3.75	4
Filtered osc 8	**4**	0	4	4	0.05	1.6	2.5	2.5
Filtered osc 16	**4**	0	4	4	0.05	0.9	1.5	2
Filtered osc 32	**4**	0	4	4	0.05	0.2	1.25	1.5
Neuron 1	**98.78**	78.68	**98.78**	23.6	51.92	51.92	52	52
Neuron 2	**10.24**	10.2	**10.24**	10.22	8.26	8.28	8.3	8.4
Non-holonomic	**7.5**	0	**7.5**	**7.5**	0	0.02	0.1	0.2
Rod reactor	**50**	0	**50**	**50**	0.1	0.2	16.6	27
Switching	**0.1**	0	0.1	0.1	0.01	0.1	0.1	0.1
Two tanks	**2**	0	2	2	0.01	2	2	2
Three vehicle	12	12	**12**	**12**	**12**	**12**	**12**	**12**

Table 7.15: Experimental results for different processing methods.

Abbreviations: Id : identity preconditioning, Pa : parallelepiped preconditioning, QR : QR preconditioning, Non : no processing, Sw1: shrink wrapping at every time step, Sw2: shrink wrapping after 2 time steps, Sw5: shrink wrapping after 5 time steps, Sw10: shrink wrapping after 10 time steps.

the same amount of X and Y. The reaction rate is influenced by the concentration of the chemicals and if one of the concentrations approaches zero the reaction rate will also approach zero. That is, the dashed lines depicted on the left side of the Figure 7.22 will contract (eventually becoming points), but since lower concentration also results in lower reaction rates, the shape will go through the crescent shape depicted on the right.

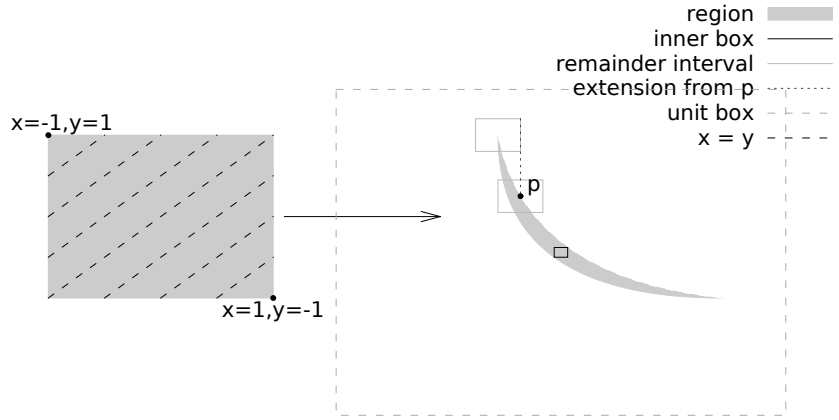


Figure 7.22: System evolving into crescent shape.

This crescent shape is unfavourable for shrink wrapping due to 2 reasons:

(i) small inner box,

(ii) long extension from some points.

(i) is not favourable since the inner box will be scaled to the unit box and if the inner box is as much smaller than the unit box then this will result in a large scaling factor.

(ii) shrink wrapping is also proportional to the value (5.6). When the crescent shape is padded with interval some points have long extensions.

Chapter 8

Conclusion

The main motivation for this thesis was to find an angle in applying computer science to synthetic biology. We decided on trying to analyse systems using formal methods. For this goal, validated integration seemed to be a particularly good fit. However, during our initial experiments, we found out that applying validated integration can be of a too hard of a task on some systems. Since working on improving the method itself seemed like a too hard of a task, we chose to instead pick a different angle in applying it. More precisely, we chose to focus on the compositional nature often found in the engineered (biological) systems.

In this thesis, we have developed and presented a compositional view of the Taylor model based validated integration. We have presented the characterisation of the systems that we consider to be compositional. In that setting, we have explained how parts of the system can be viewed as (partially) independent systems - components. We have adapted the theory and algorithms of the Taylor model based validated integration to be compositional (i.e. centred around components).

We have done the same thing for the Taylor model based validated integration that uses processing methods between the integration steps.

For method with shrink wrapping, we have explained how integration can be compositional and why shrink wrapping cannot be compositional.

For method with preconditioning, we have explained how preconditioning can be compositional and given criteria for a particular preconditioning method being compositional. Additionally, when a preconditioning method is compositional, we have detailed and presented different degrees of composition that can be used with it. That is, compositional integration together with compositional preconditioning or compositional integration together with non-compositional preconditioning.

We have implemented the compositional algorithms in our tool CFlow*. Using our tool we have experimented on 29 systems. We have done 4 kinds of experiments:

1. finding the best compositional preconditioning method,
2. the effect on performance when using compositional algorithms compared to non-compositional one,
3. performance gains from using alternative implementation for intervals,
4. the feasibility of shrink wrapping.

From these experiments, 1 and 4 provide a way to compare different processing methods (in general we have found it to be lacking in the literature). In 4, we have seen and reasoned about why shrink wrapping behaves poorly compared to preconditioning.

From 2, we have seen that depending on the system, the performance gains of using compositional algorithms can vary greatly. We have also seen it vary in different parts of the algorithm. We looked more closely at the interval refinement part of the algorithm and explained it can produce the biggest performance gains on some of the systems.

During experiments, we found out that composition does not provide as big of performance gains as we initially hoped for. We explained this in 3 and provided an alternative implementation to one of the core classes of Flow* that results in significant performance gain (together with and without the compositional approach).

We find our results in some ways promising and in some ways disappointing. We have seen that using the compositional approach can yield performance gains, but not on the level that previously unsolvable systems could be solved.

Information about the availability of the tool CFlow* and means to reproduce the data for experiments used in this thesis is given in Appendix A.

8.1 Going Forward

The work present in this thesis can be taken forward in the following ways

Relaxing restrictions. The current compositional setting imposes restrictions on the systems where it can be applied. One of the main restriction is that there cannot be any cyclic dependencies between components. One possible way to overcome this is to abstract the effect of how a part of the system affects another part with an interval box.

This is the approach what is taken in [CS16]. There might be value in an approach that does not fully abstract away from representing the effect symbolically.

Adapting compositional versions of more precondition methods. We have presented identity and parallelepiped preconditioning methods as compatible with composition. In [MB11] there is a list of other preconditioning methods, some of which could also be adapted to preconditioning.

Point initial conditions. Most often the initial conditions are represented with interval boxes which after begin converted into Taylor model introduces a parameter for each dimension. Point initial conditions do not need to introduce a parameter, but our current implementation does it. We have done preliminary work on not introducing point initial condition in our tool CFlow*.

Parameters having a small effect. Our current compositional setting only differentiates between whether a variable influences another one or not. There is likely value in quantifying this more precisely. The effect of one variable on another might be negligible and there might be value in abstracting that effect more aggressively compared to other ones.

Different time step sizes for different components. Our current approach requires that all components advance with the same time step size. This restriction could be lifted by introducing ways to combine two Taylor models into one or glueing Taylor models together to be able to use them in the Picard operator.

Compositional ODE solver in the solver for hybrid systems. Flow* makes use of their non-compositional ODE solver in their solver for hybrid systems. With some modifications, it is possible to use our CFlow* also in a solver for hybrid systems.

Appendix A

Obtaining CFlow* and Data for Experiments

Our tool CFlow* and its source code may be obtained from the repository at `https://github.com/kristjanl/fs-extension-code`.

The models files, scripts to run the tool on systems (and generate data presented in this thesis) can be found at the following directories:

- scripts for generating data for Tables 7.2, 7.3, 7.4, 7.5, 7.6 and 7.7 at `experiments/ext_sorted`,
- scripts for generating data for Tables 7.8 and 7.9 at `experiments/ext_hybrid`,
- scripts for generating data for Tables 7.10, 7.11, 7.12, 7.13 and 7.14 at `experiments/ext_dint`,
- the CFlow* models files at `models/compositional/extended`.

Appendix B

CFlow* Experiments

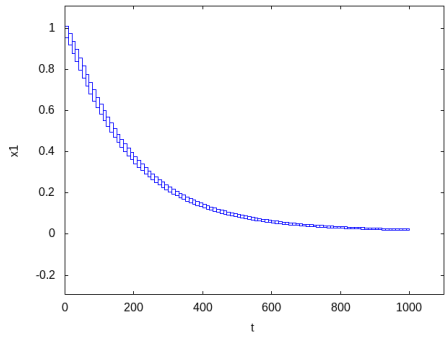
B.1 Flowpipes for the Systems

All the flowpipes computed in Section 7.3 are presented in Figures B.1 to B.29. The missing graphs indicate that the method was inapplicable.

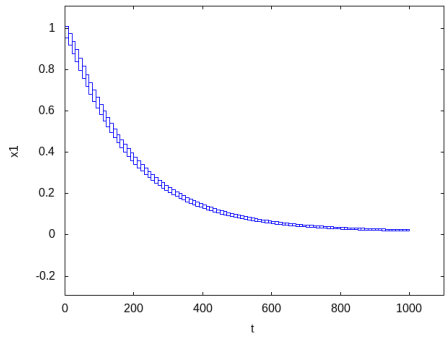
B.2 Data for Processing Methods

In this section we present data for comparing different processing methods. We will compare:

- preconditioning methods
  - identity preconditioning
  - parallelepiped preconditioning
  - QR preconditioning
- no processing between integration phases
- shrink wrapping
  - at every step
  - after every 2 steps
  - after every 5 steps
  - after every 10 steps

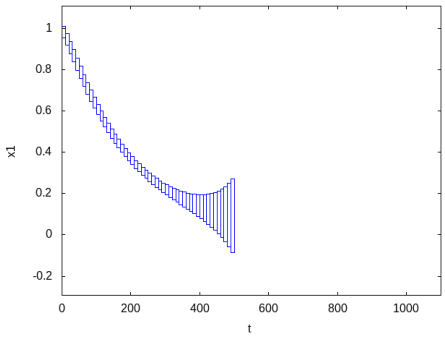


(a) Identity preconditioning

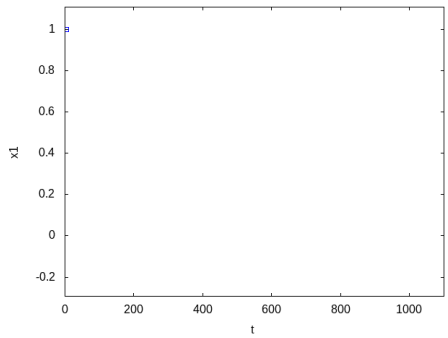


(b) Parallelepiped preconditioning

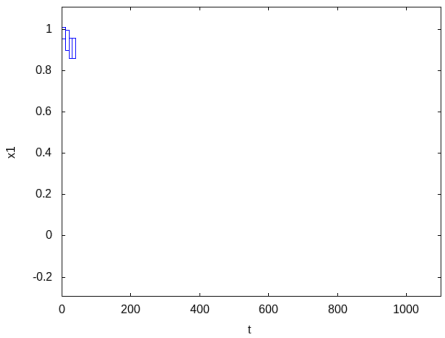
(c) QR preconditioning



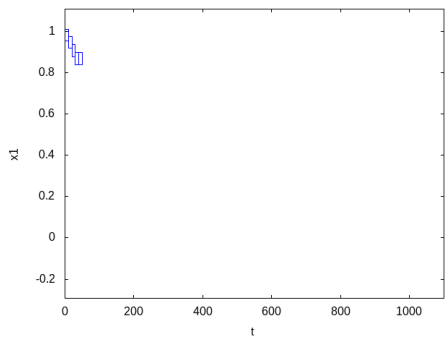
(d) No processing



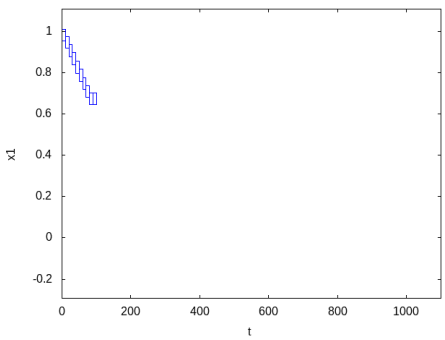
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

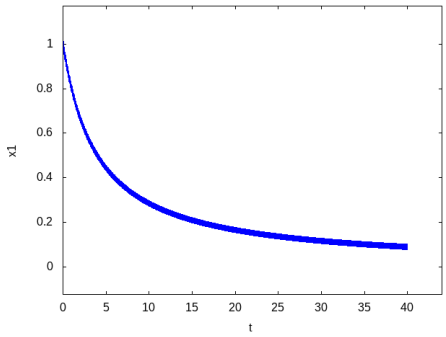


(g) Shrink wrapping after 5 time steps

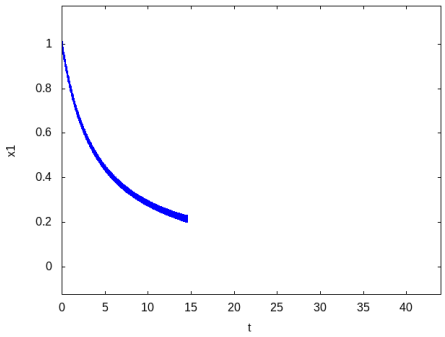


(h) Shrink wrapping after 10 time steps

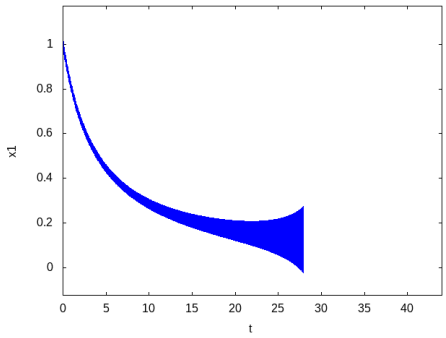
Figure B.1: Flowpipes for AND-Gate system using different processing methods.



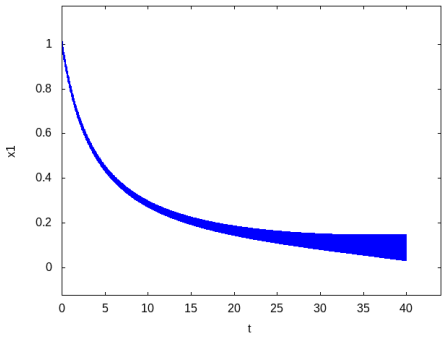
(a) Identity preconditioning



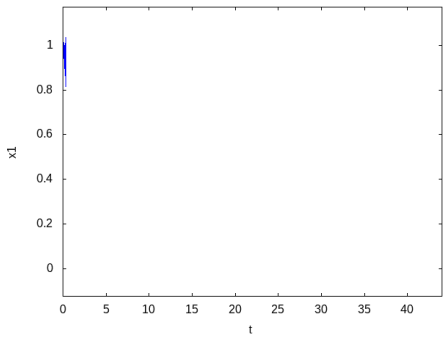
(b) Parallelepiped preconditioning



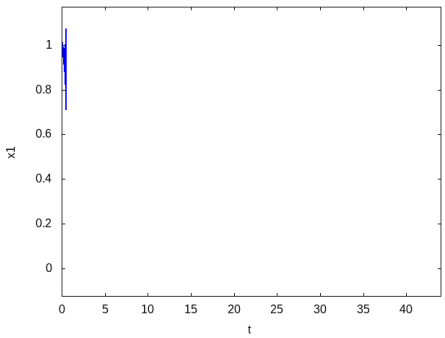
(c) QR preconditioning



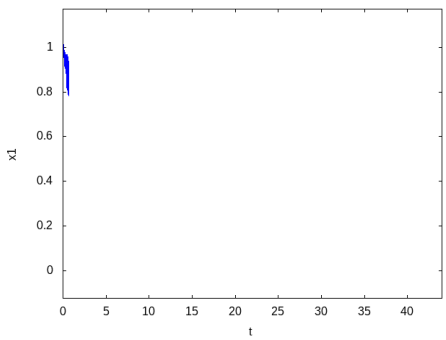
(d) No processing



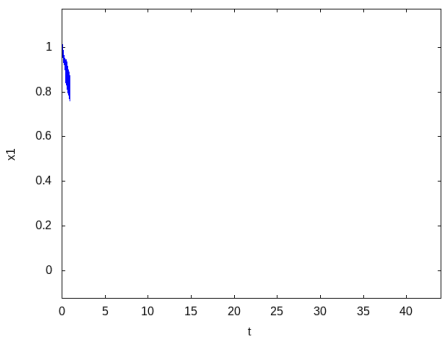
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

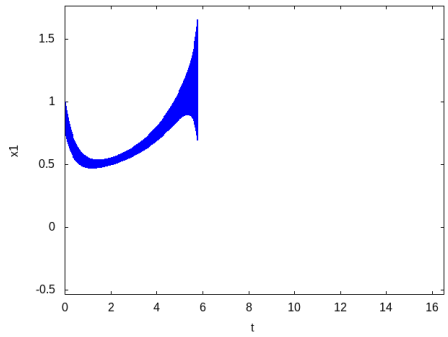


(g) Shrink wrapping after 5 time steps

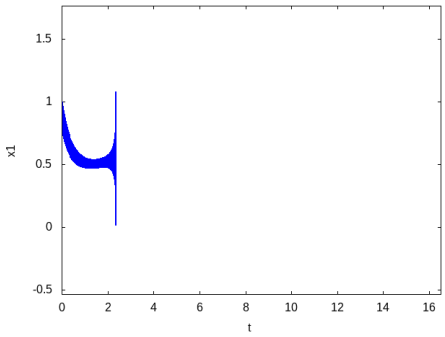


(h) Shrink wrapping after 10 time steps

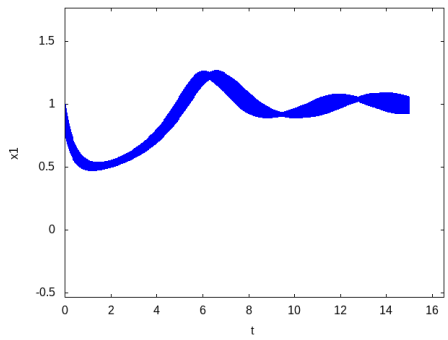
Figure B.2: Flowpipes for AND-OR Gate system using different processing methods.



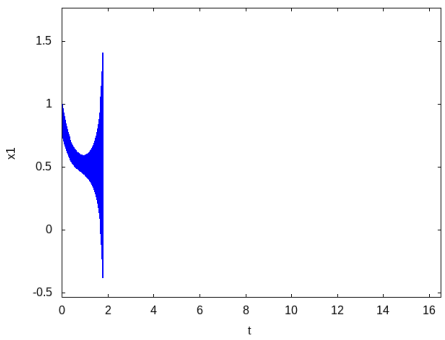
(a) Identity preconditioning



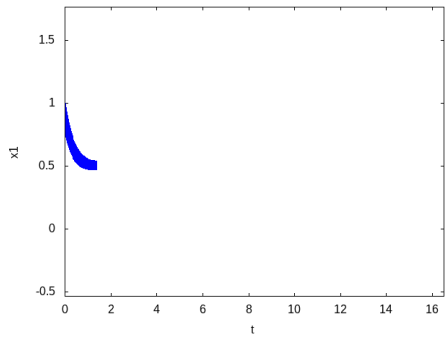
(b) Parallelepiped preconditioning



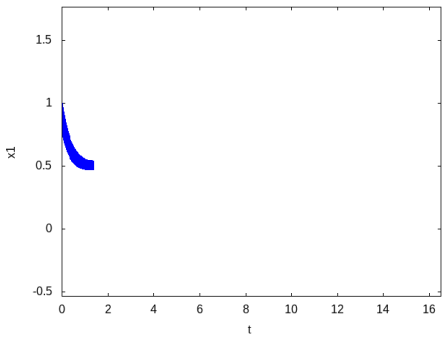
(c) QR preconditioning



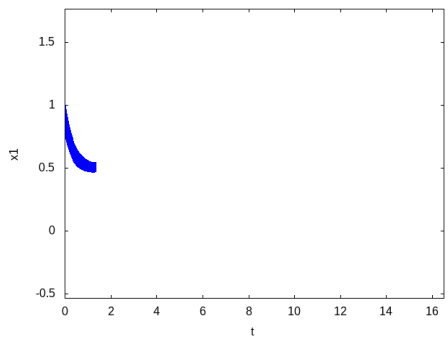
(d) No processing



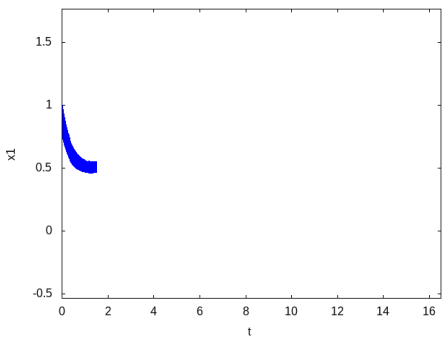
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

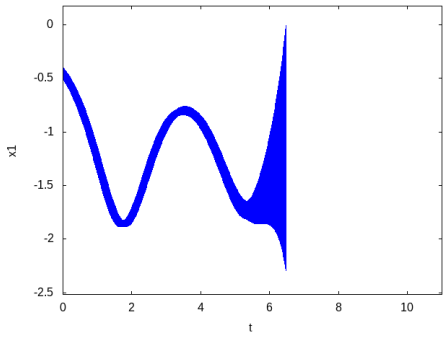


(g) Shrink wrapping after 5 time steps

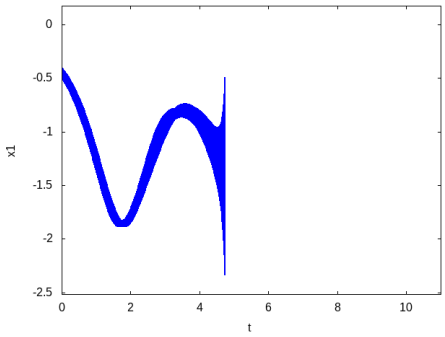


(h) Shrink wrapping after 10 time steps

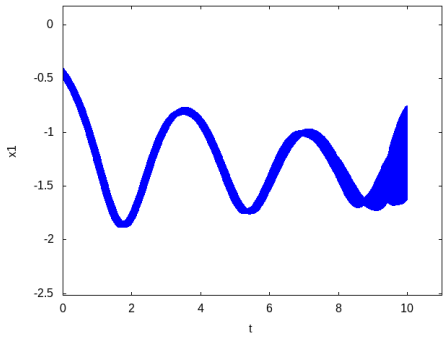
Figure B.3: Flowpipes for Brusselator system using different processing methods.



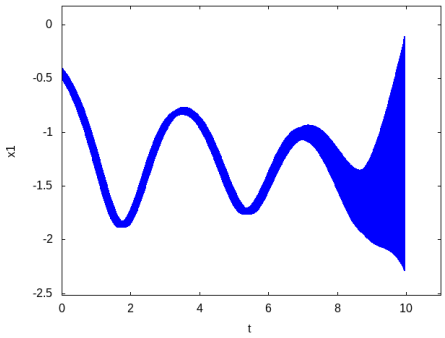
(a) Identity preconditioning



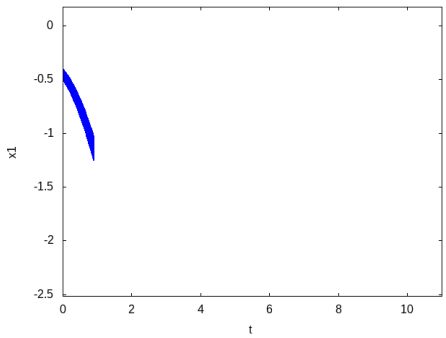
(b) Parallelepiped preconditioning



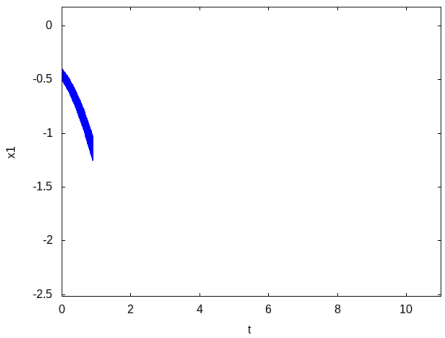
(c) QR preconditioning



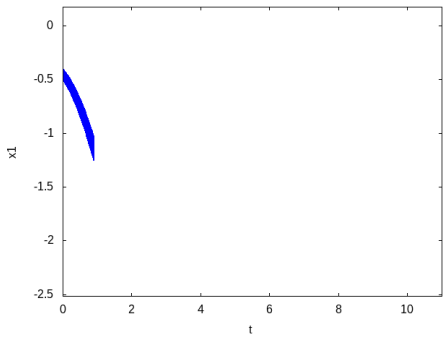
(d) No processing



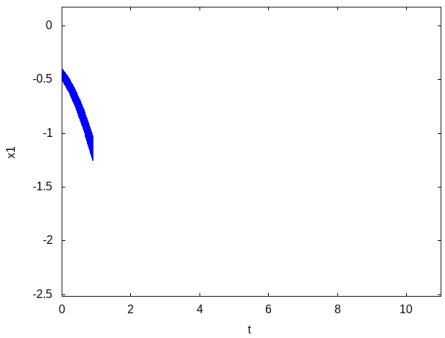
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

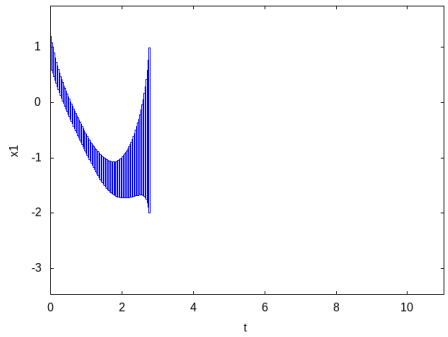


(g) Shrink wrapping after 5 time steps

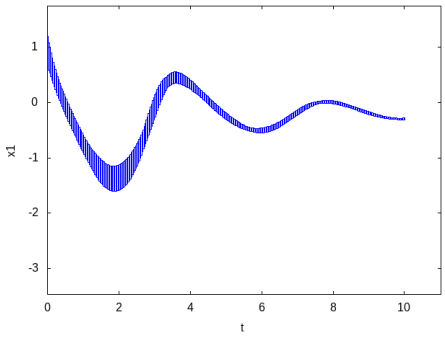


(h) Shrink wrapping after 10 time steps

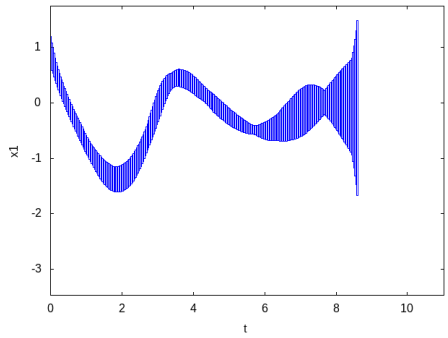
Figure B.4: Flowpipes for Buckling col system using different processing methods.



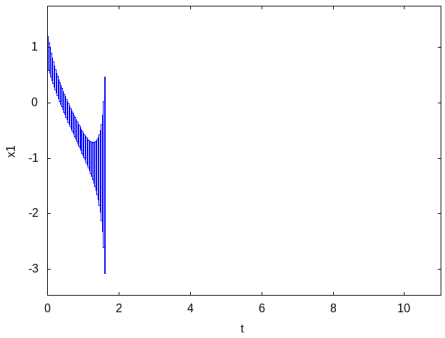
(a) Identity preconditioning



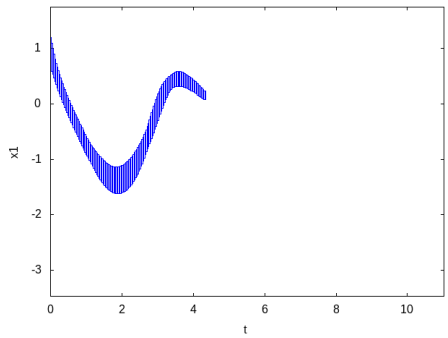
(b) Parallelepiped preconditioning



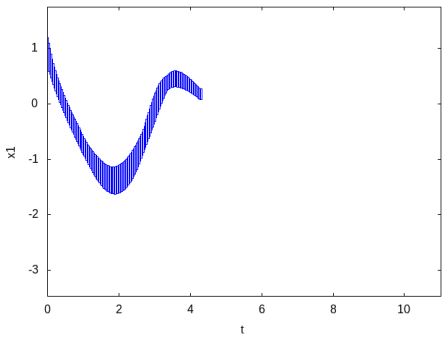
(c) QR preconditioning



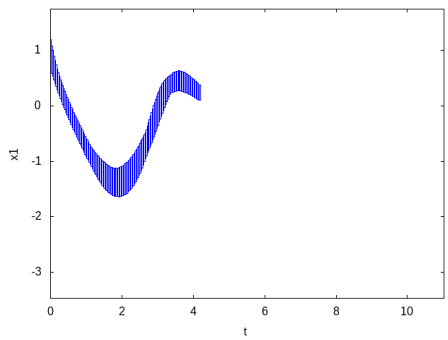
(d) No processing



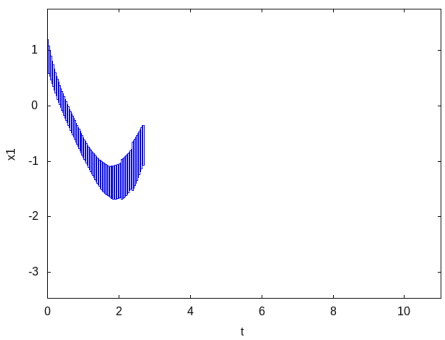
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

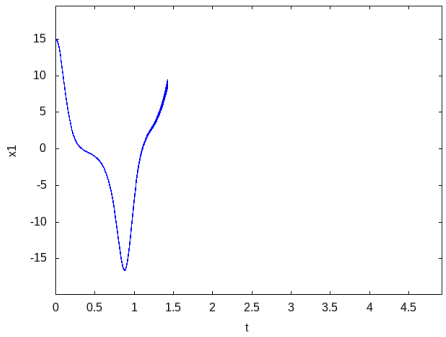


(g) Shrink wrapping after 5 time steps

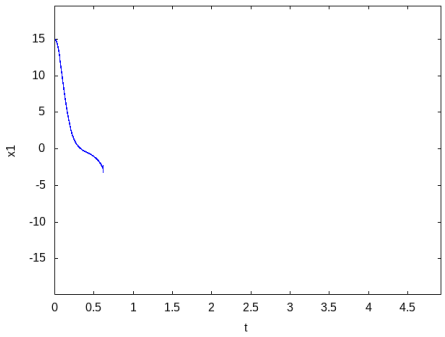


(h) Shrink wrapping after 10 time steps

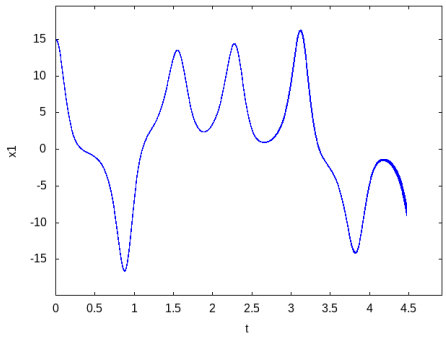
Figure B.5: Flowpipes for Jet engine system using different processing methods.



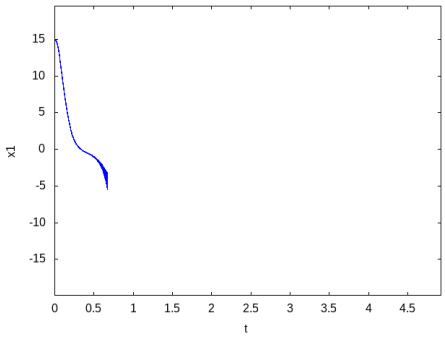
(a) Identity preconditioning



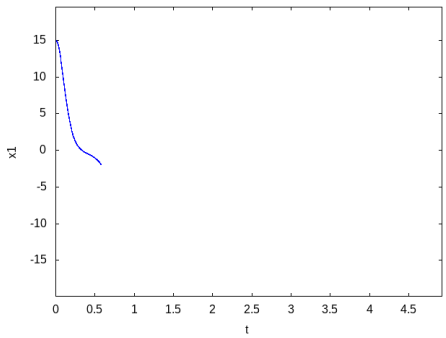
(b) Parallelepiped preconditioning



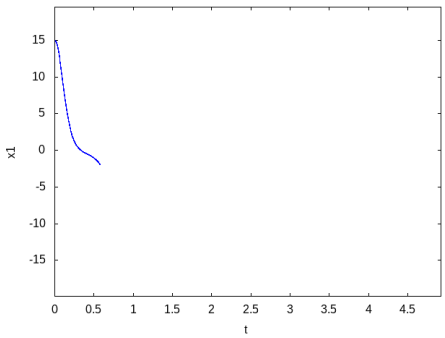
(c) QR preconditioning



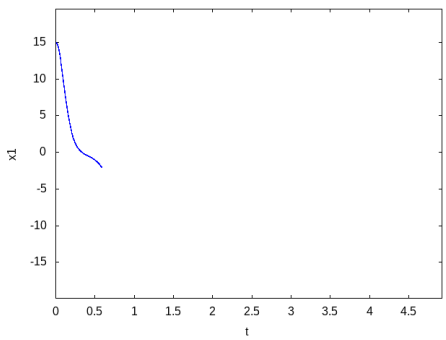
(d) No processing



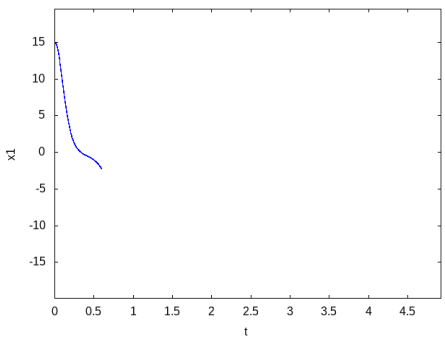
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

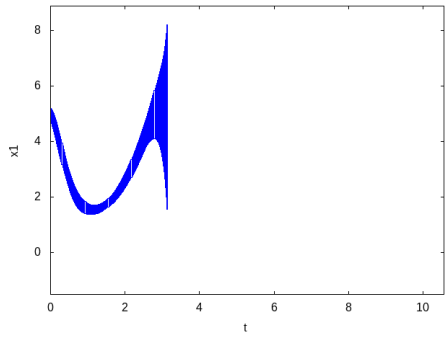


(g) Shrink wrapping after 5 time steps

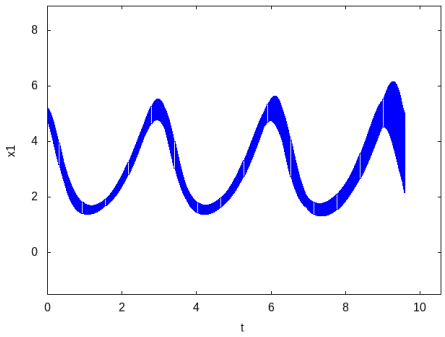


(h) Shrink wrapping after 10 time steps

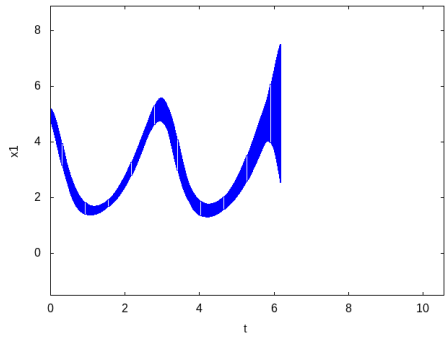
Figure B.6: Flowpipes for Lorentz system using different processing methods.



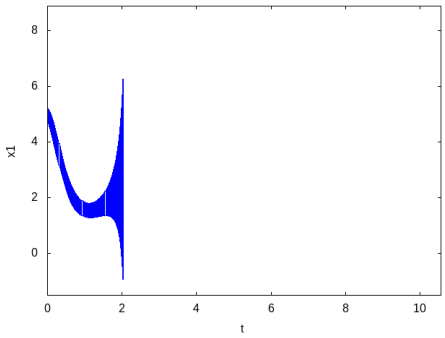
(a) Identity preconditioning



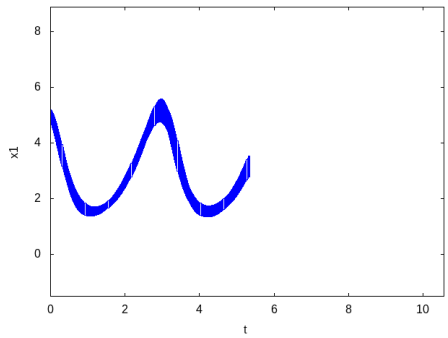
(b) Parallelepiped preconditioning



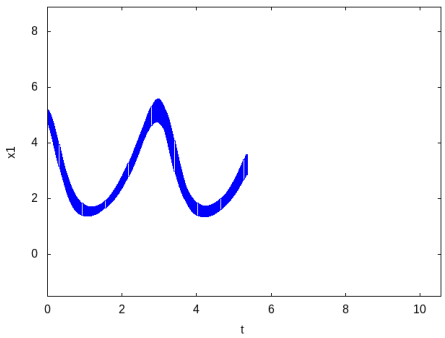
(c) QR preconditioning



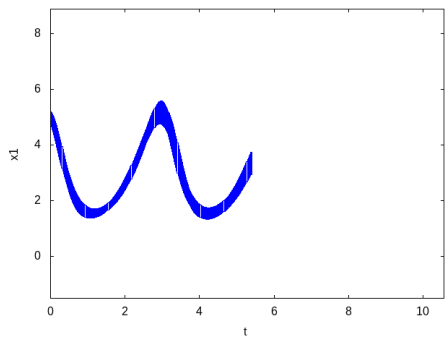
(d) No processing



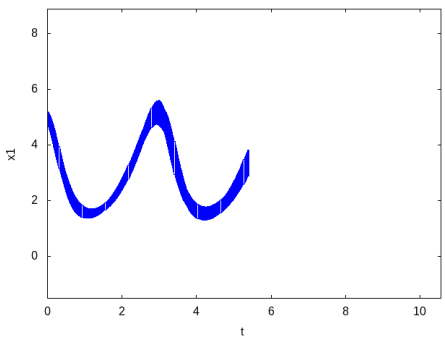
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

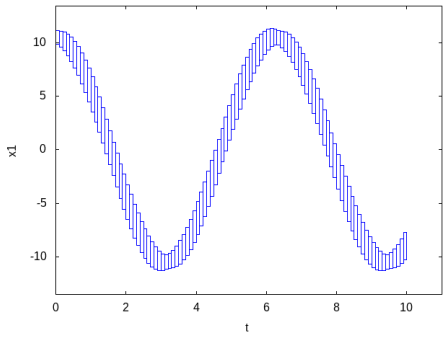


(g) Shrink wrapping after 5 time steps

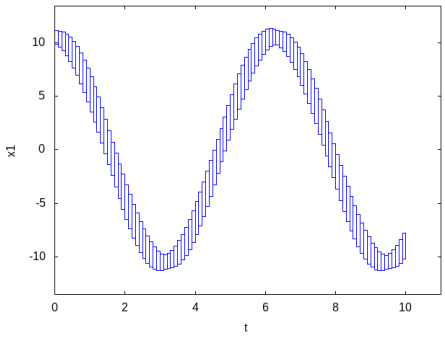


(h) Shrink wrapping after 10 time steps

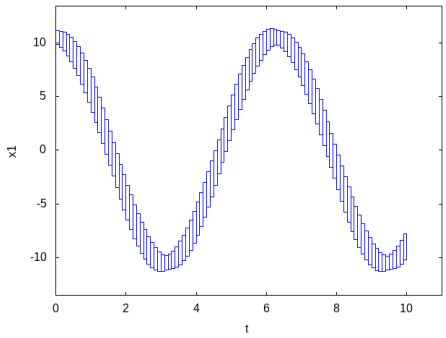
Figure B.7: Flowpipes for Lotka-Volterra system using different processing methods.



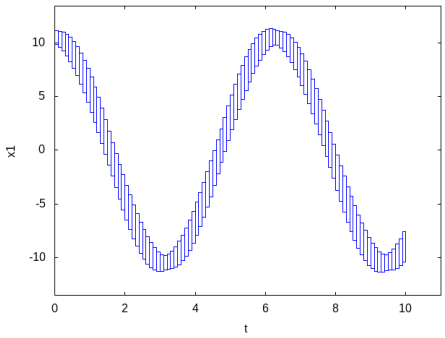
(a) Identity preconditioning



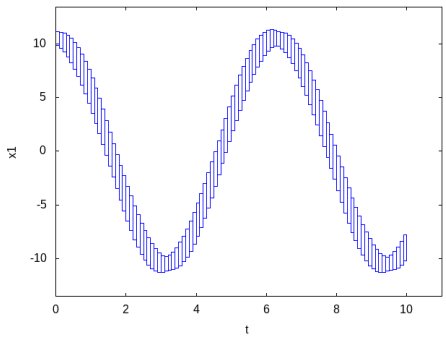
(b) Parallelepiped preconditioning



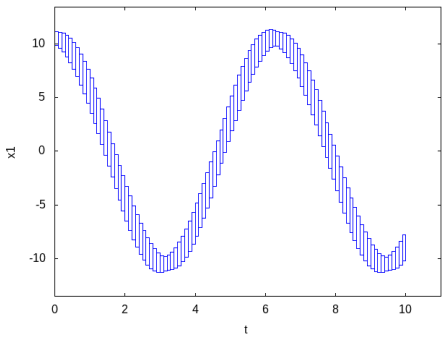
(c) QR preconditioning



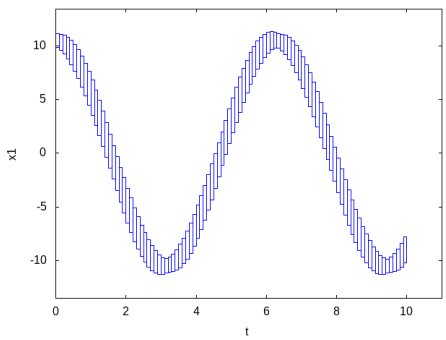
(d) No processing



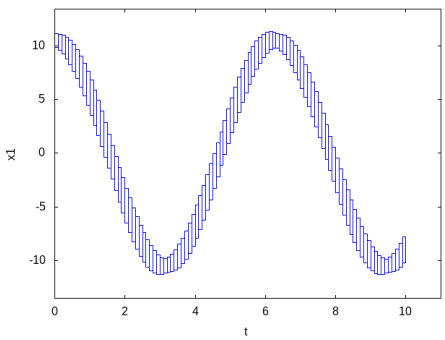
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

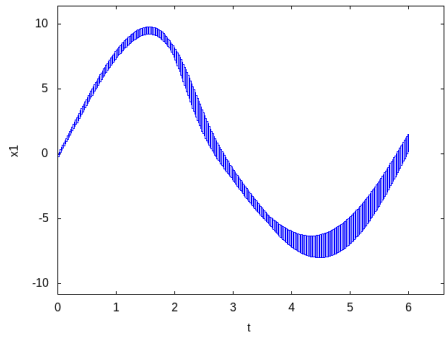


(g) Shrink wrapping after 5 time steps



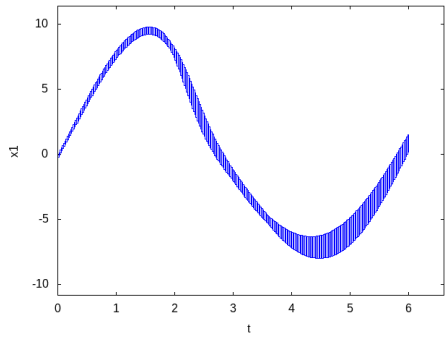
(h) Shrink wrapping after 10 time steps

Figure B.8: Flowpipes for Moore rot system using different processing methods.

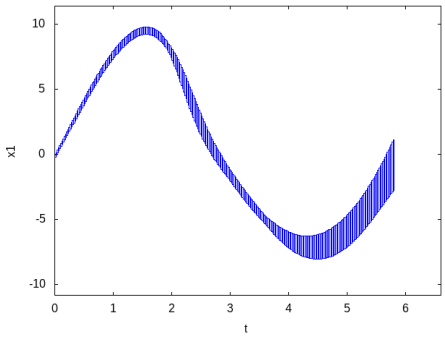


(a) Identity preconditioning

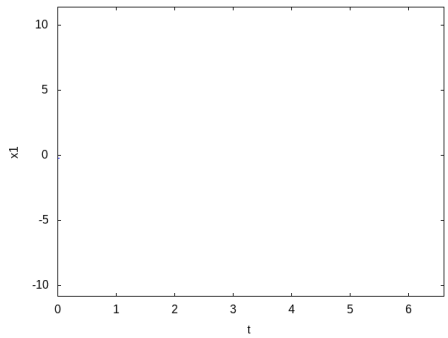
(b) Parallelepiped preconditioning



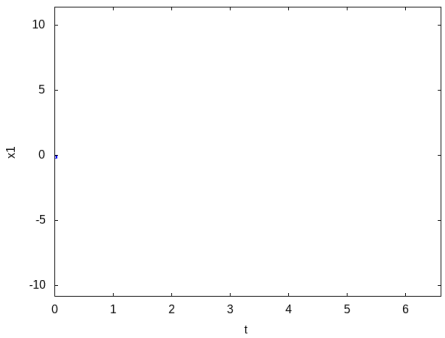
(c) QR preconditioning



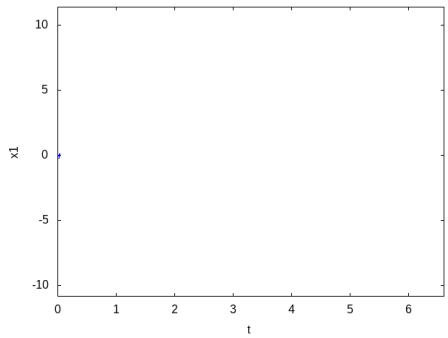
(d) No processing



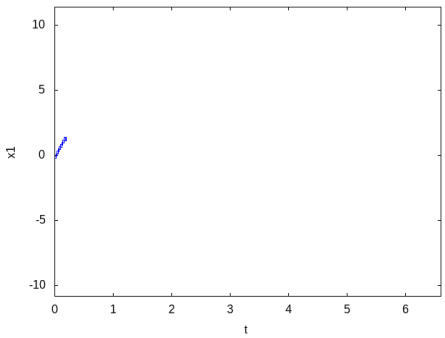
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

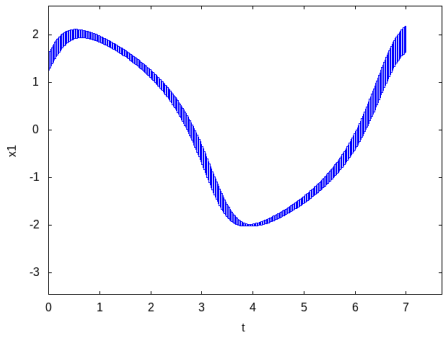


(g) Shrink wrapping after 5 time steps

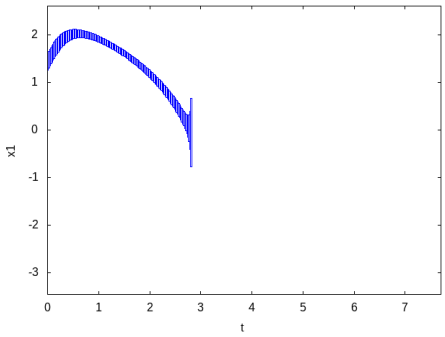


(h) Shrink wrapping after 10 time steps

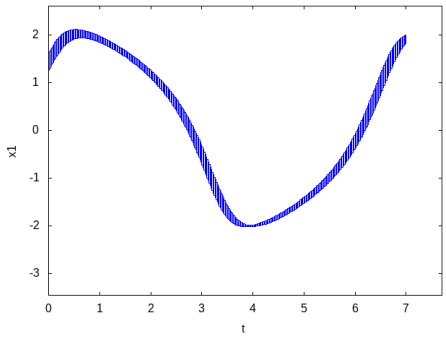
Figure B.9: Flowpipes for Roessler system using different processing methods.



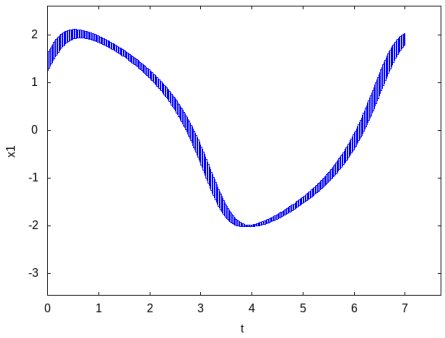
(a) Identity preconditioning



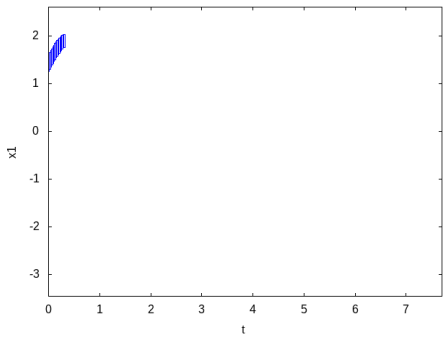
(b) Parallelepiped preconditioning



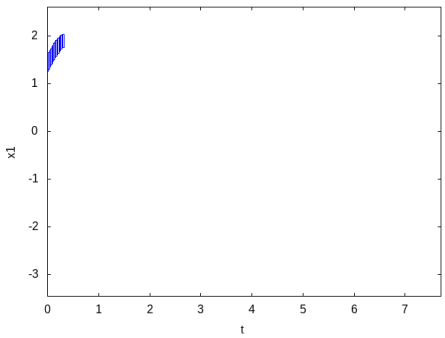
(c) QR preconditioning



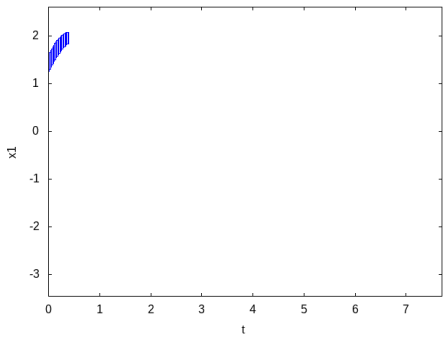
(d) No processing



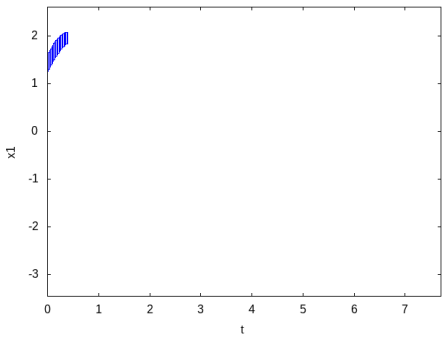
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

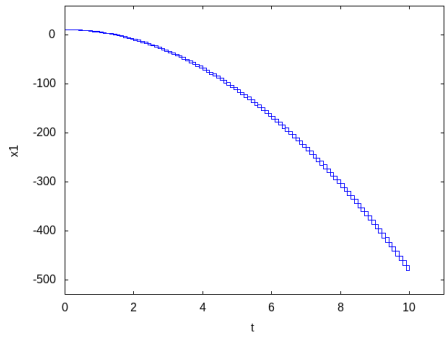


(g) Shrink wrapping after 5 time steps

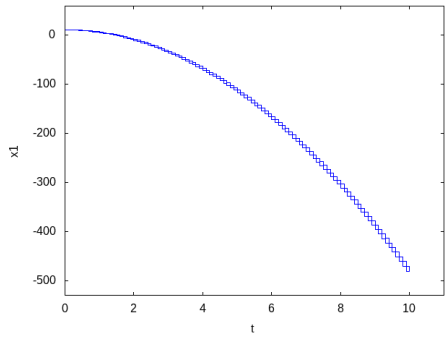


(h) Shrink wrapping after 10 time steps

Figure B.10: Flowpipes for Vanderpol system using different processing methods.

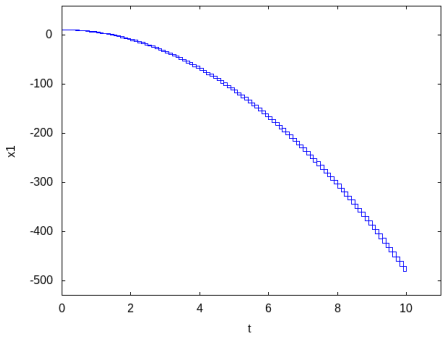


(a) Identity preconditioning

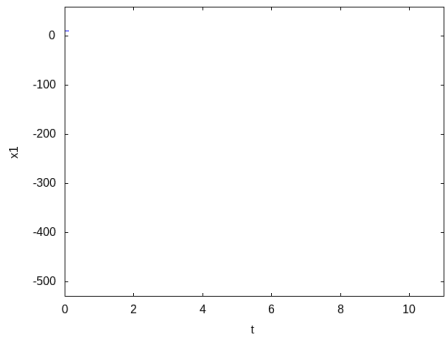


(b) Parallelepiped preconditioning

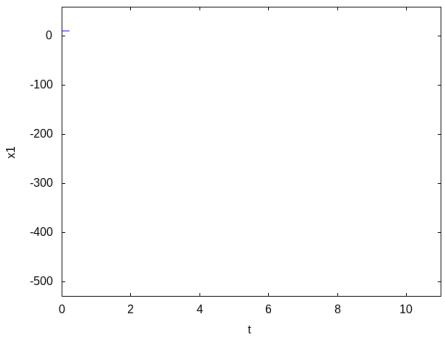
(c) QR preconditioning



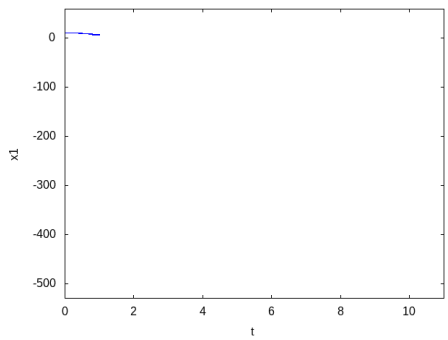
(d) No processing



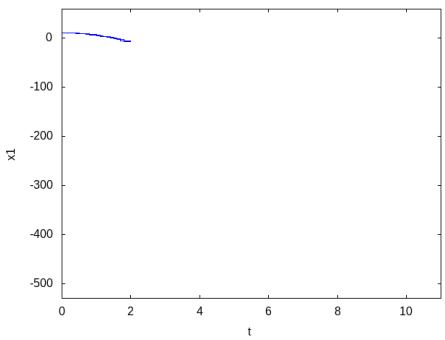
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

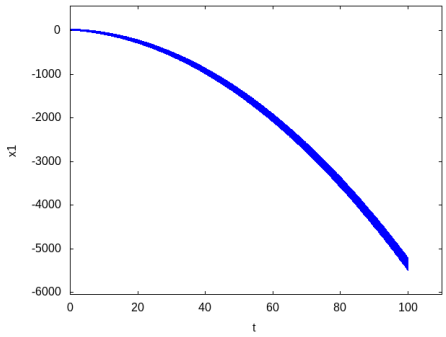


(g) Shrink wrapping after 5 time steps

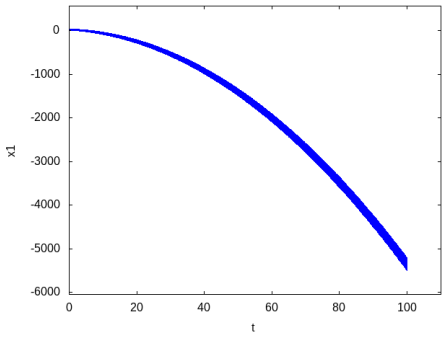


(h) Shrink wrapping after 10 time steps

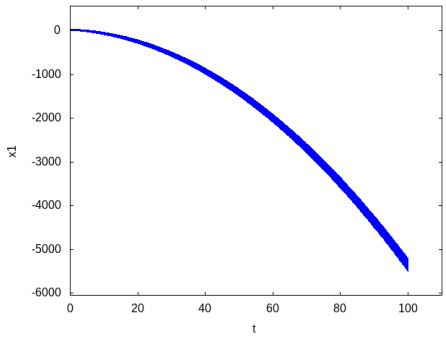
Figure B.11: Flowpipes for Bouncing ball system using different processing methods.



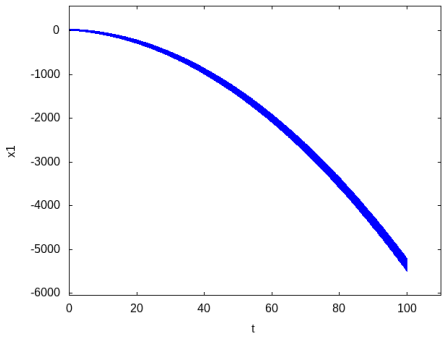
(a) Identity preconditioning



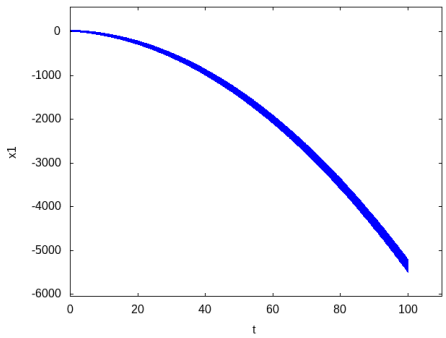
(b) Parallelepiped preconditioning



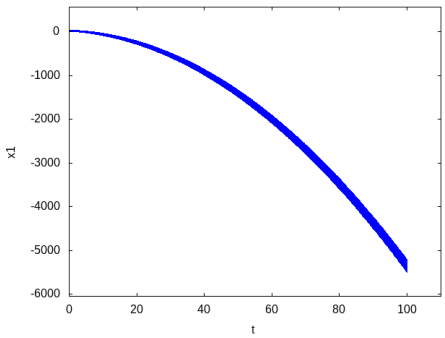
(c) QR preconditioning



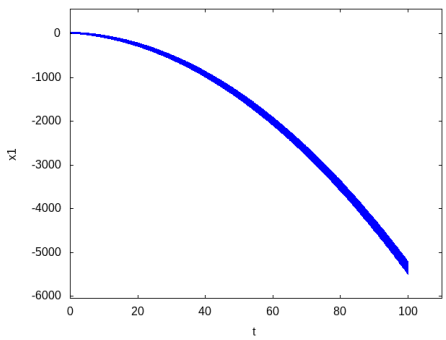
(d) No processing



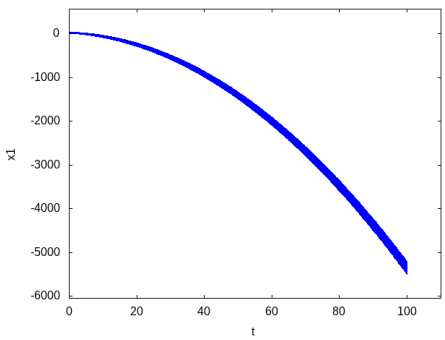
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

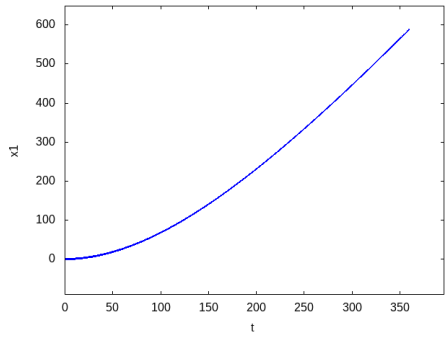


(g) Shrink wrapping after 5 time steps

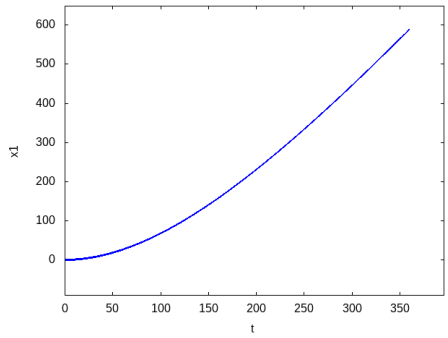


(h) Shrink wrapping after 10 time steps

Figure B.12: Flowpipes for Cruise control system using different processing methods.

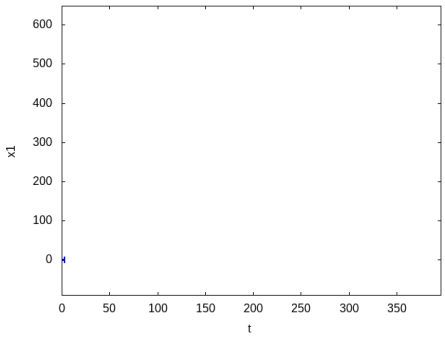


(a) Identity preconditioning

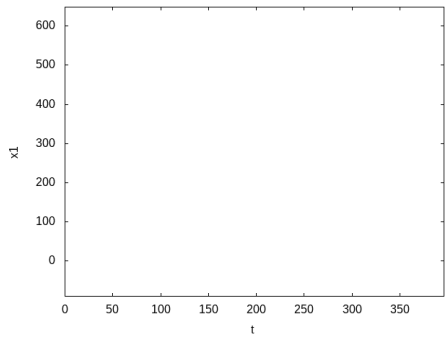


(b) Parallelepiped preconditioning

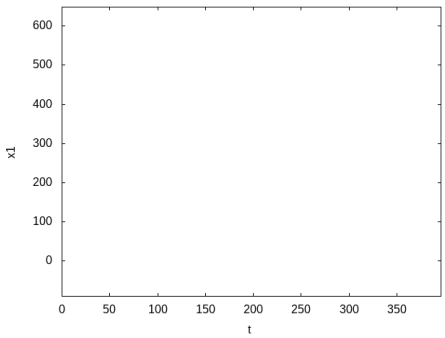
(c) QR preconditioning



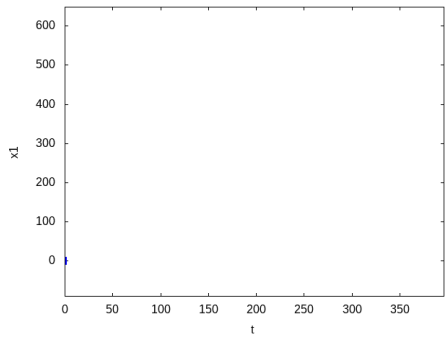
(d) No processing



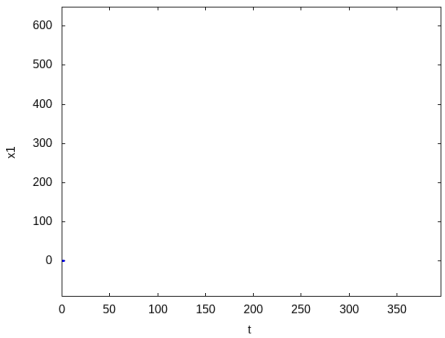
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

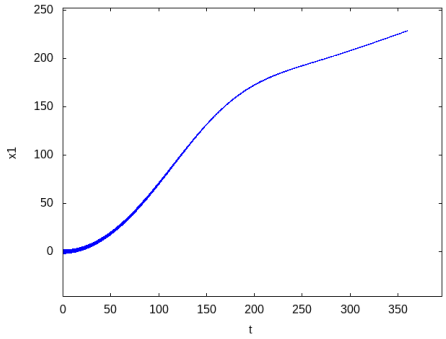


(g) Shrink wrapping after 5 time steps



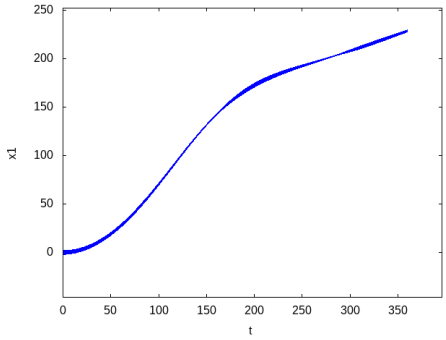
(h) Shrink wrapping after 10 time steps

Figure B.13: Flowpipes for Glycemic 1 system using different processing methods.

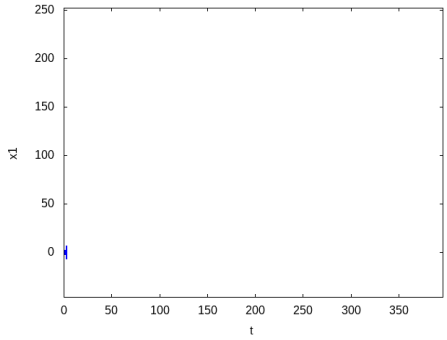


(a) Identity preconditioning

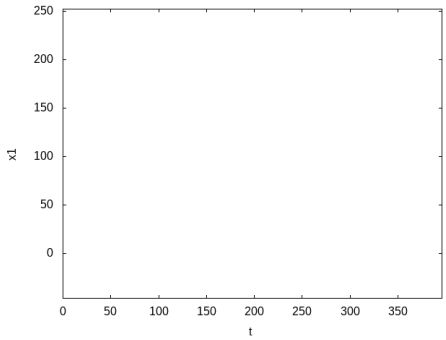
(b) Parallelepiped preconditioning



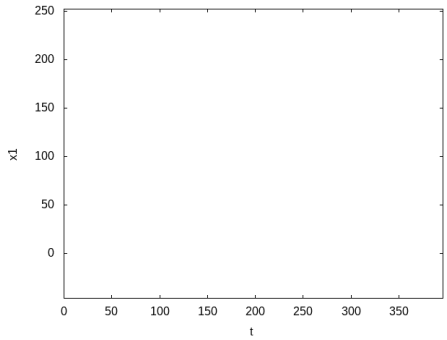
(c) QR preconditioning



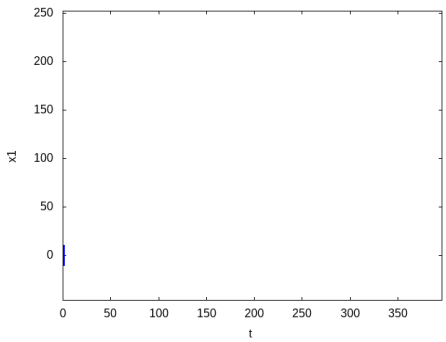
(d) No processing



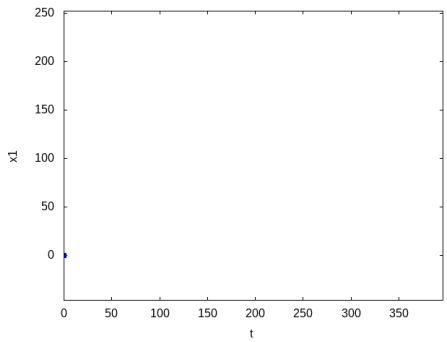
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

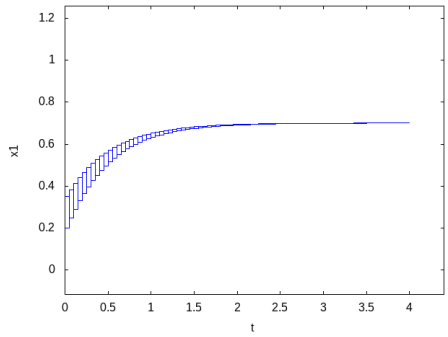


(g) Shrink wrapping after 5 time steps



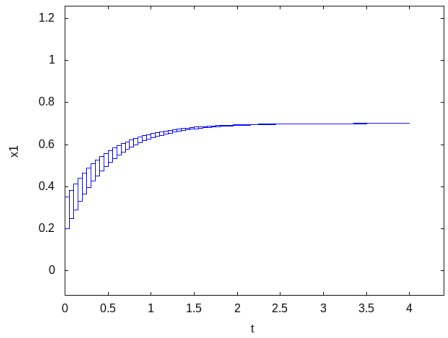
(h) Shrink wrapping after 10 time steps

Figure B.14: Flowpipes for Glycemic 2 system using different processing methods.

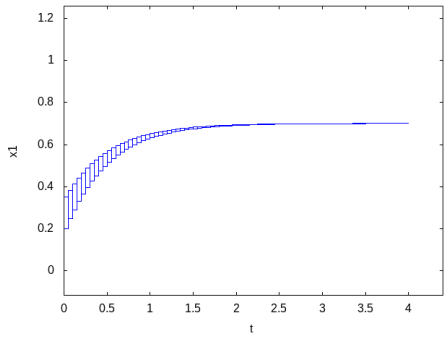


(a) Identity preconditioning

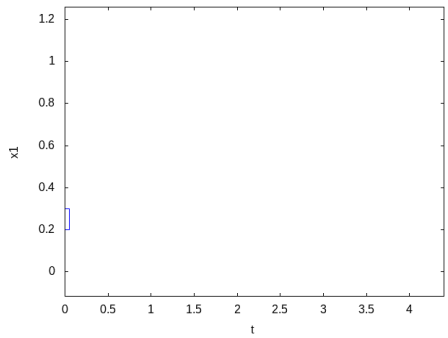
(b) Parallelepiped preconditioning



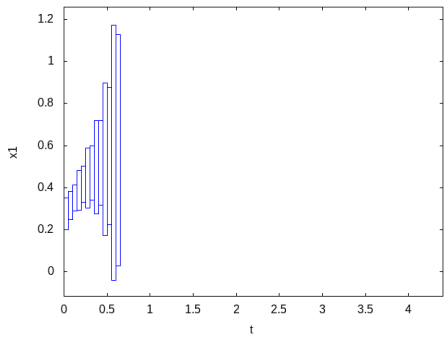
(c) QR preconditioning



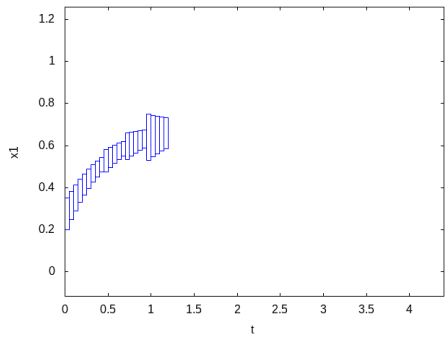
(d) No processing



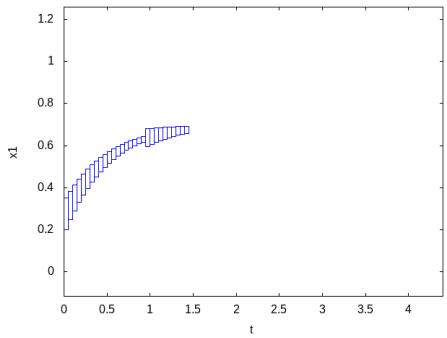
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

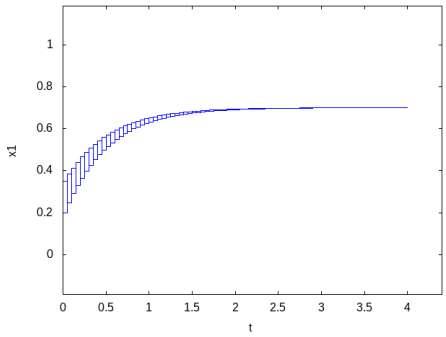


(g) Shrink wrapping after 5 time steps



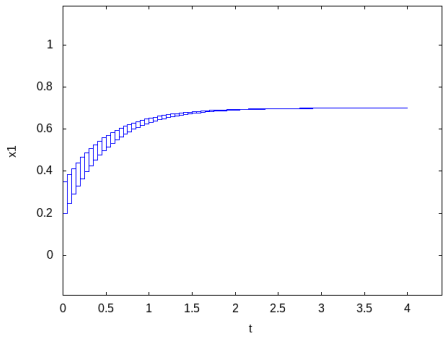
(h) Shrink wrapping after 10 time steps

Figure B.15: Flowpipes for Filtered osc 4 system using different processing methods.

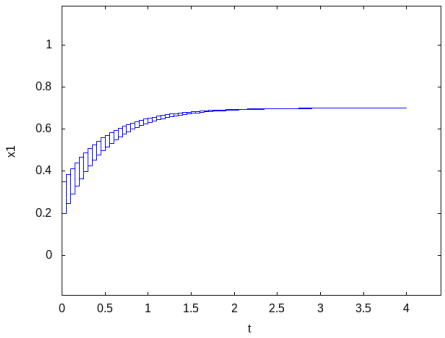


(a) Identity preconditioning

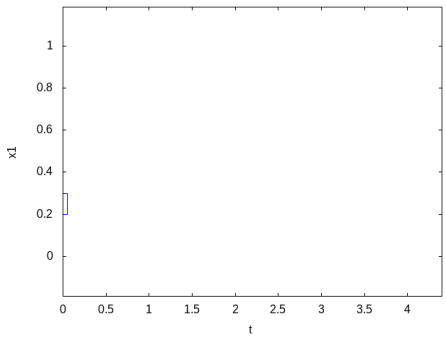
(b) Parallelepiped preconditioning



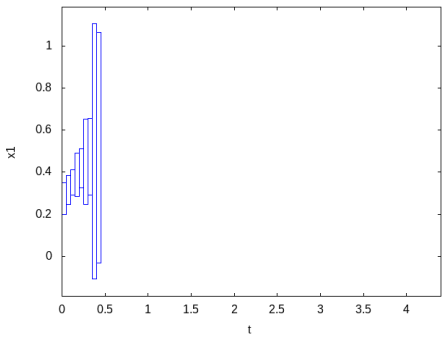
(c) QR preconditioning



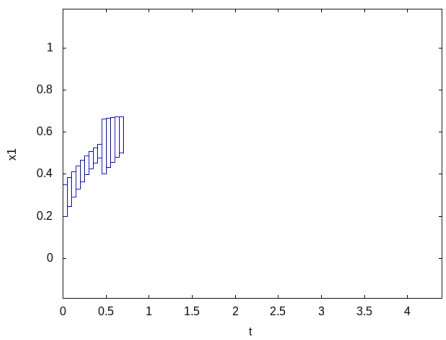
(d) No processing



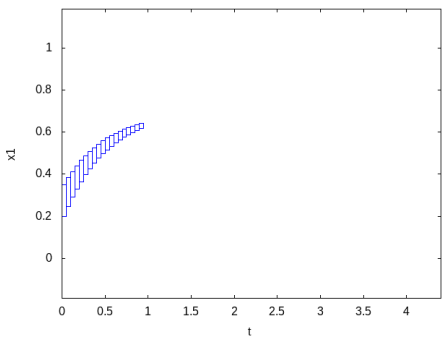
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

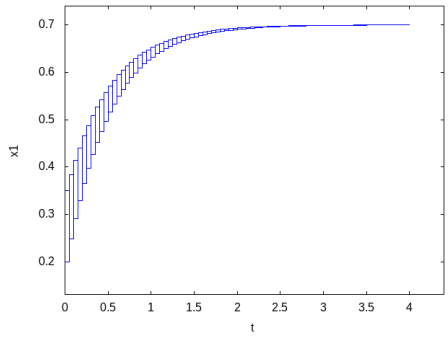


(g) Shrink wrapping after 5 time steps



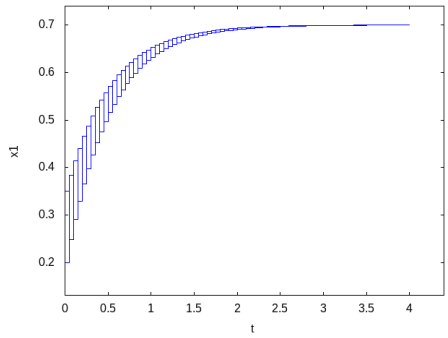
(h) Shrink wrapping after 10 time steps

Figure B.16: Flowpipes for Filtered osc 8 system using different processing methods.

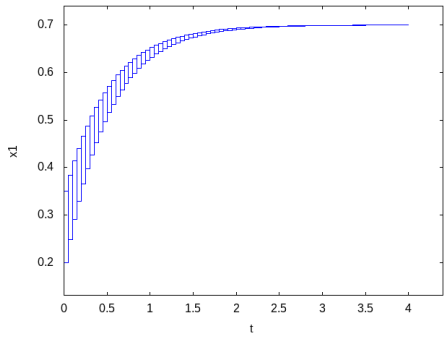


(a) Identity preconditioning

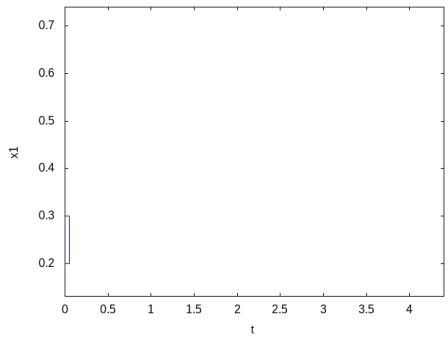
(b) Parallelepiped preconditioning



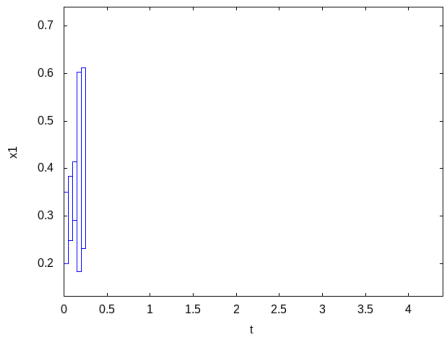
(c) QR preconditioning



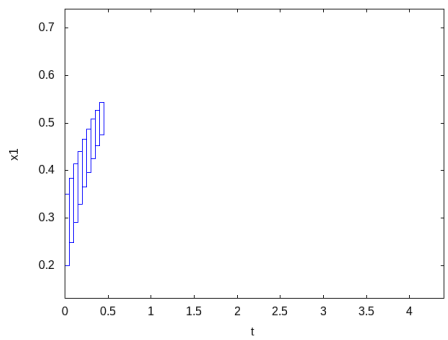
(d) No processing



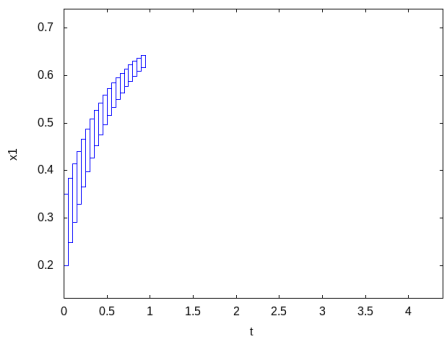
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

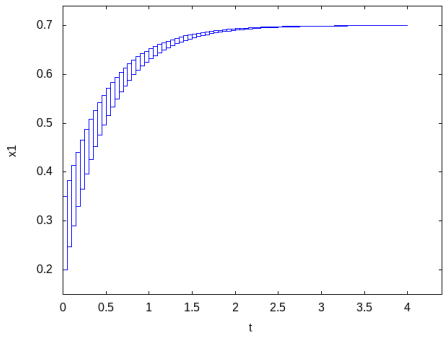


(g) Shrink wrapping after 5 time steps



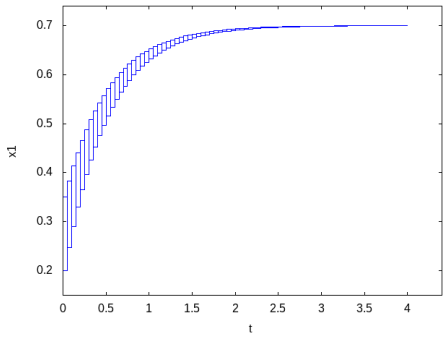
(h) Shrink wrapping after 10 time steps

Figure B.17: Flowpipes for Filtered osc 16 system using different processing methods.

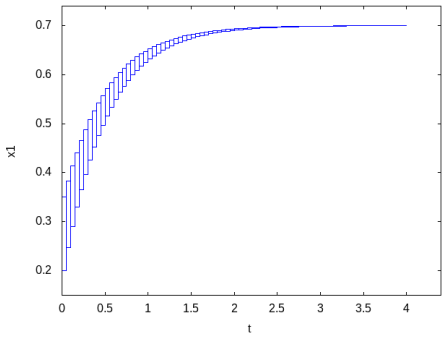


(a) Identity preconditioning

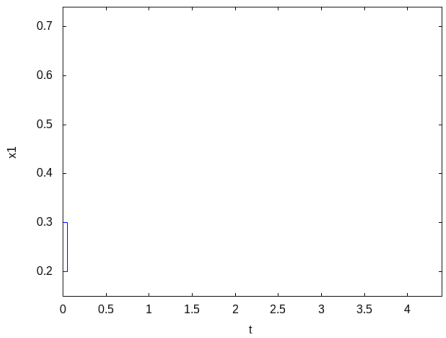
(b) Parallelepiped preconditioning



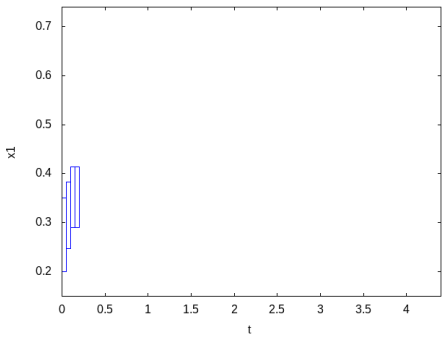
(c) QR preconditioning



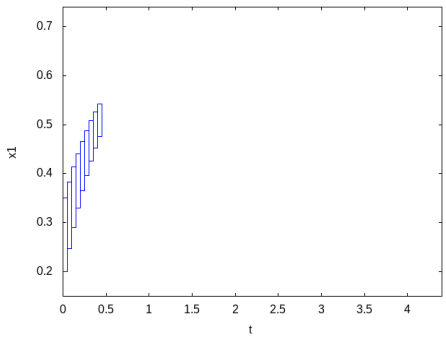
(d) No processing



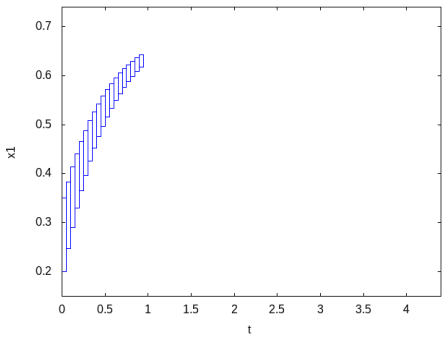
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

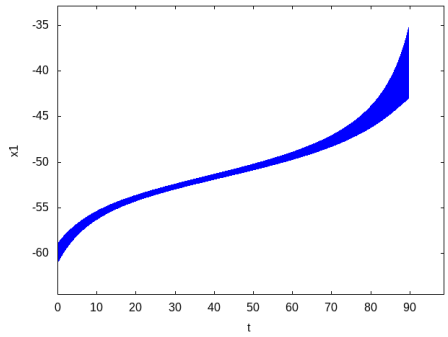


(g) Shrink wrapping after 5 time steps

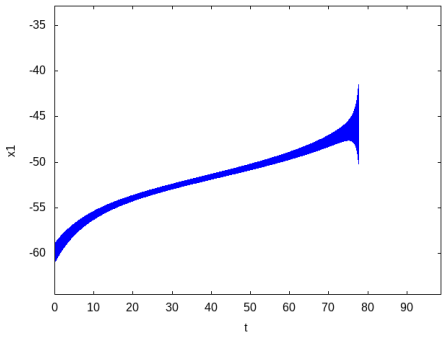


(h) Shrink wrapping after 10 time steps

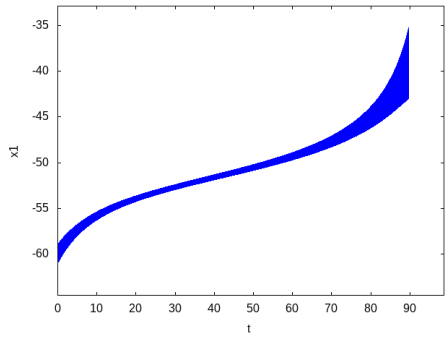
Figure B.18: Flowpipes for Filtered osc 32 system using different processing methods.



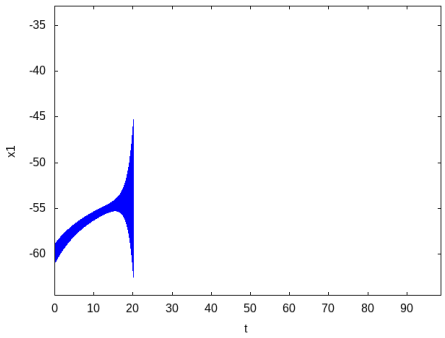
(a) Identity preconditioning



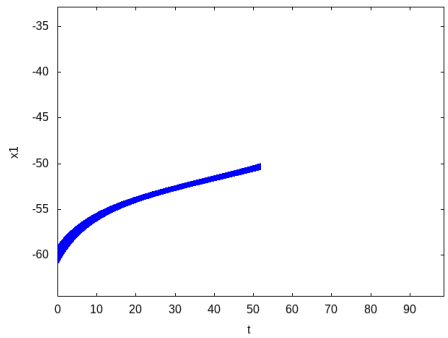
(b) Parallelepiped preconditioning



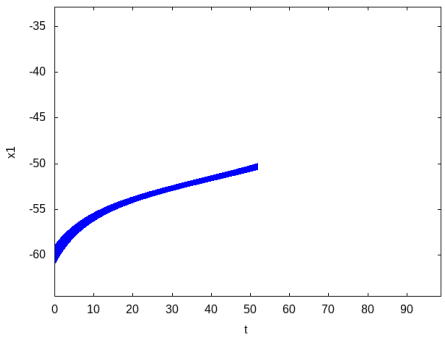
(c) QR preconditioning



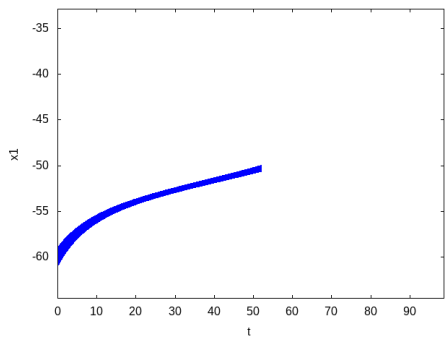
(d) No processing



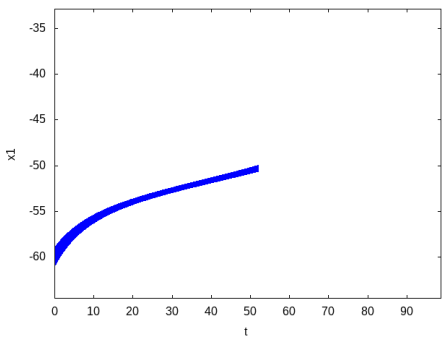
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

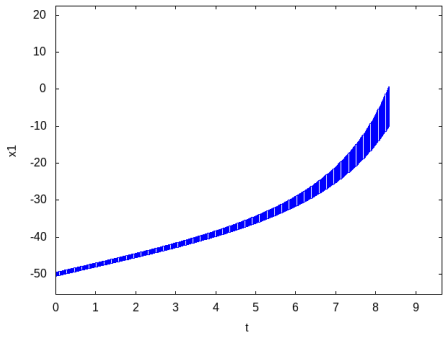


(g) Shrink wrapping after 5 time steps

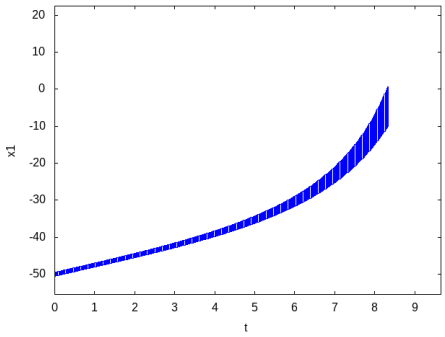


(h) Shrink wrapping after 10 time steps

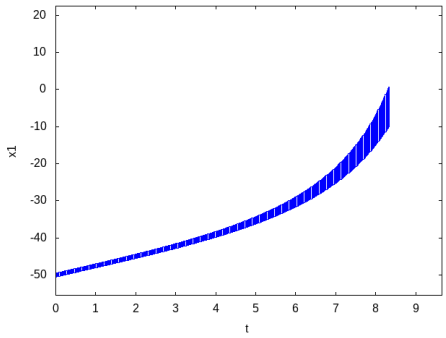
Figure B.19: Flowpipes for Neuron 1 system using different processing methods.



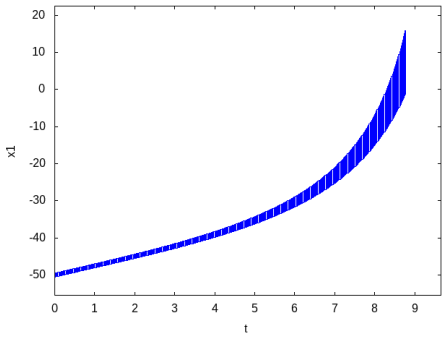
(a) Identity preconditioning



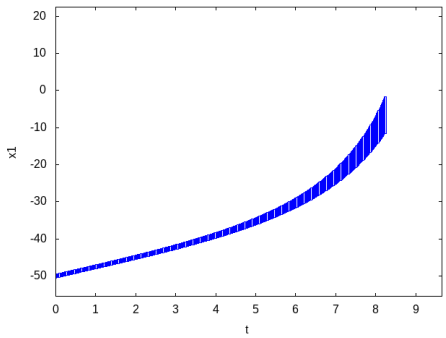
(b) Parallelepiped preconditioning



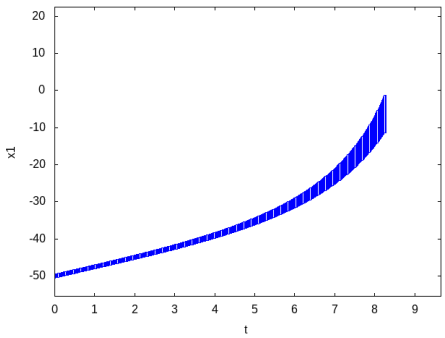
(c) QR preconditioning



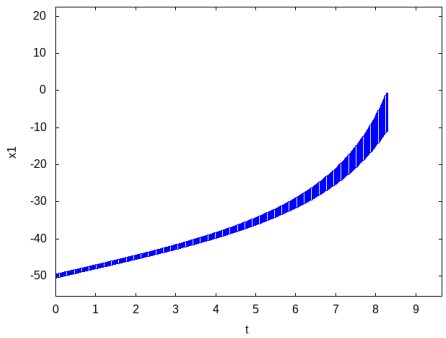
(d) No processing



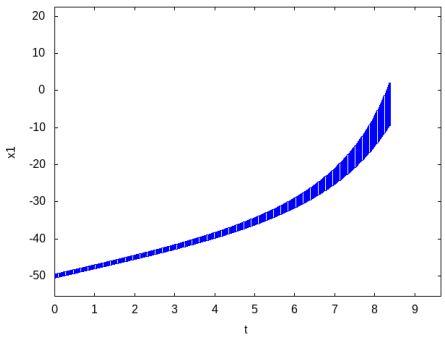
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

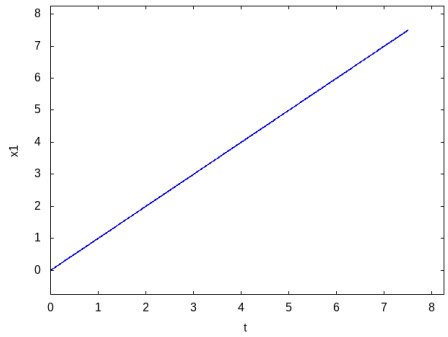


(g) Shrink wrapping after 5 time steps



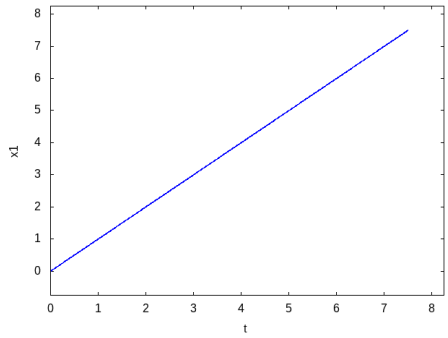
(h) Shrink wrapping after 10 time steps

Figure B.20: Flowpipes for Neuron 2 system using different processing methods.

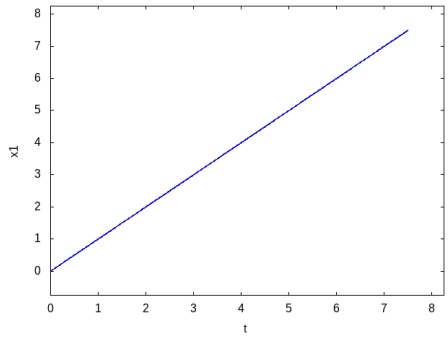


(a) Identity preconditioning

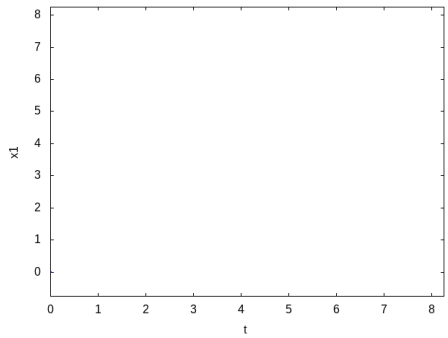
(b) Parallelepiped preconditioning



(c) QR preconditioning

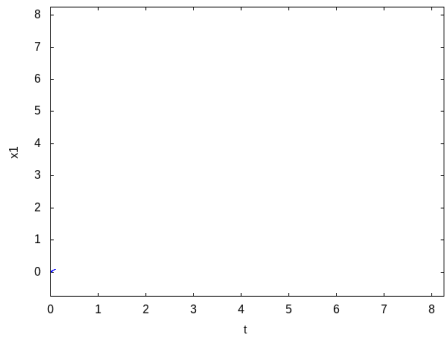


(d) No processing

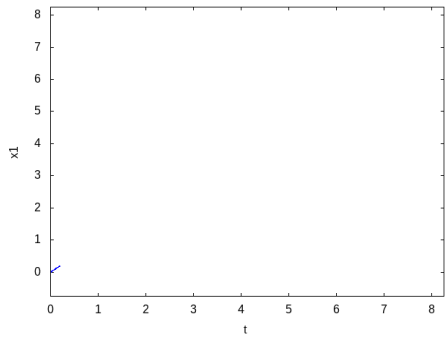


(e) Shrink wrapping at every time step

(f) Shrink wrapping after 2 time steps

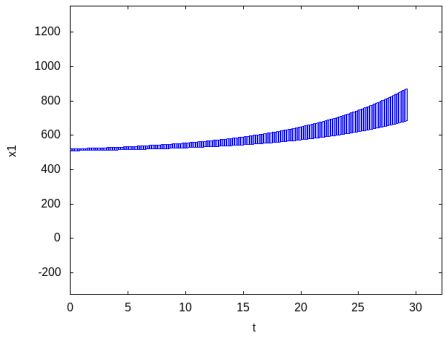


(g) Shrink wrapping after 5 time steps



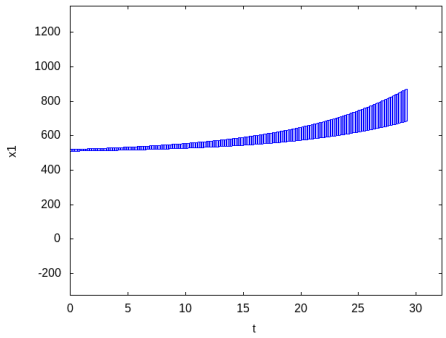
(h) Shrink wrapping after 10 time steps

Figure B.21: Flowpipes for Non-holonomic system using different processing methods.

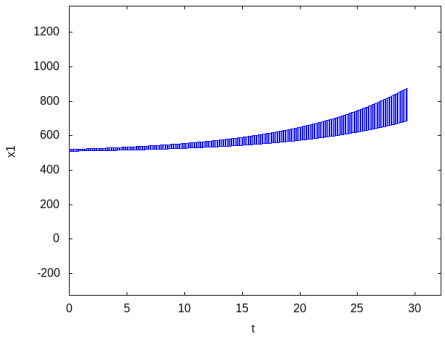


(a) Identity preconditioning

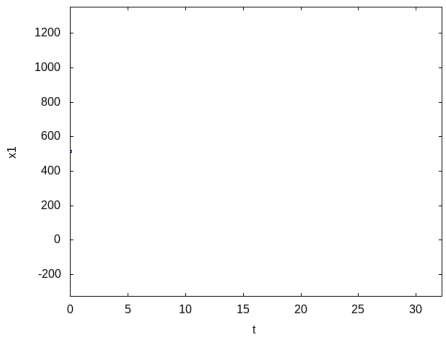
(b) Parallelepiped preconditioning



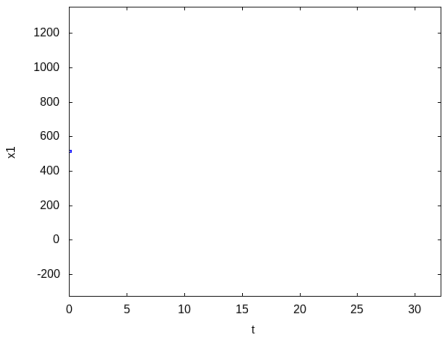
(c) QR preconditioning



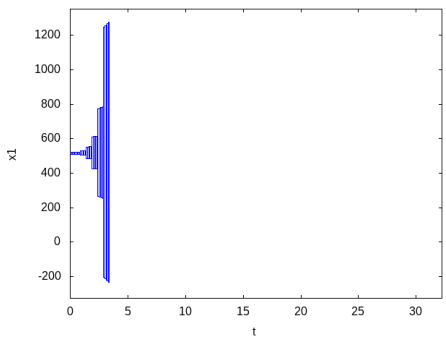
(d) No processing



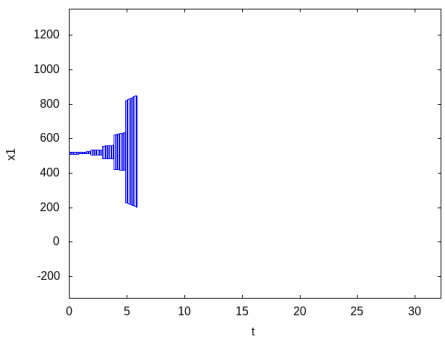
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

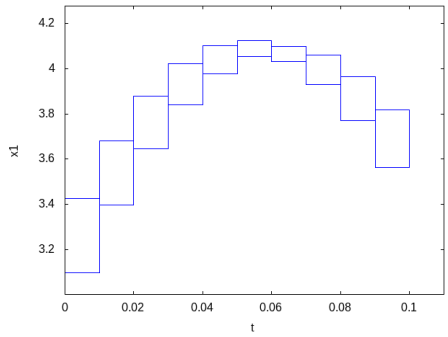


(g) Shrink wrapping after 5 time steps

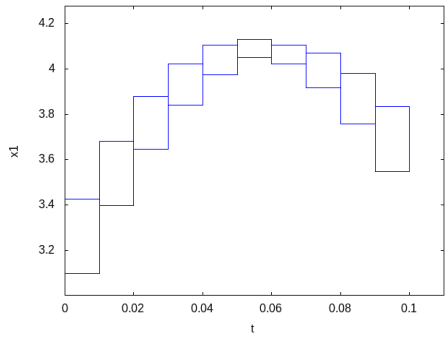


(h) Shrink wrapping after 10 time steps

Figure B.22: Flowpipes for Rod reactor system using different processing methods.

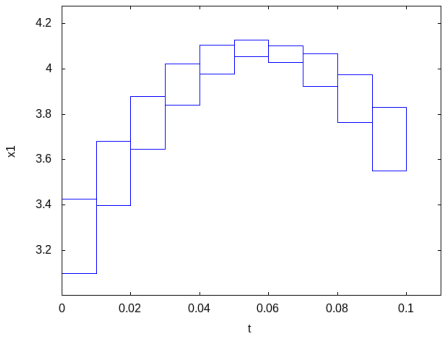


(a) Identity preconditioning

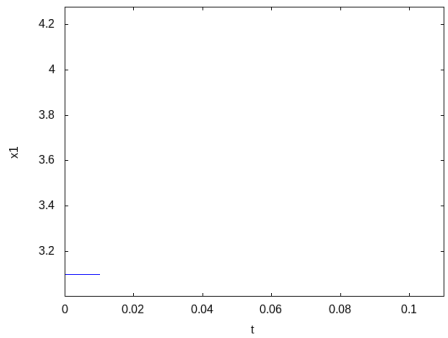


(b) Parallelepiped preconditioning

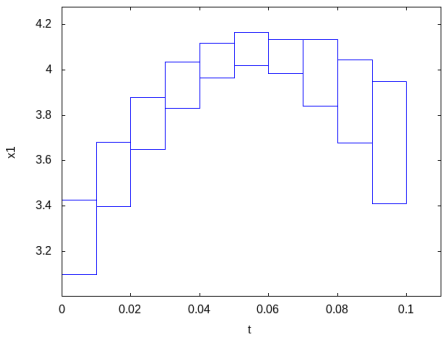
(c) QR preconditioning



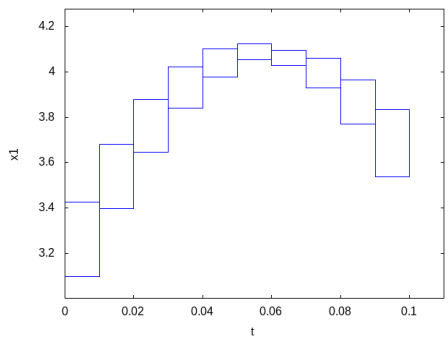
(d) No processing



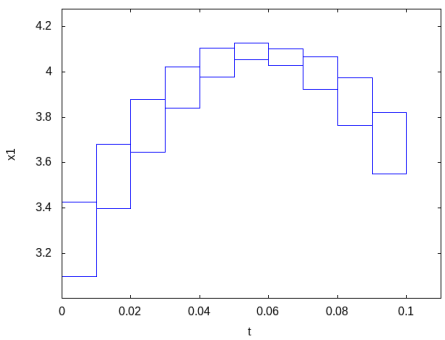
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

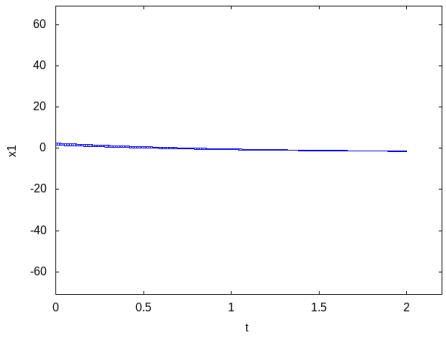


(g) Shrink wrapping after 5 time steps



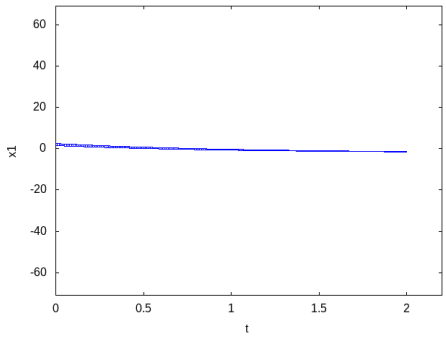
(h) Shrink wrapping after 10 time steps

Figure B.23: Flowpipes for Switching system using different processing methods.

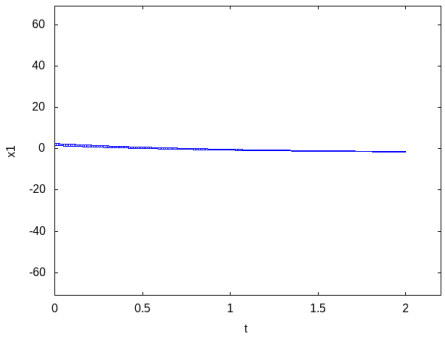


(a) Identity preconditioning

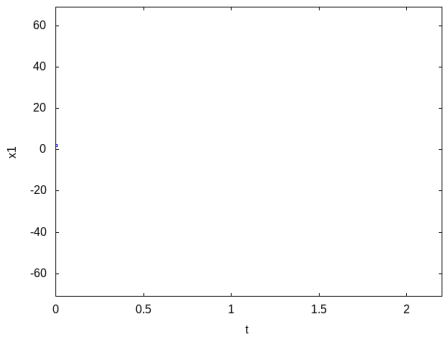
(b) Parallelepiped preconditioning



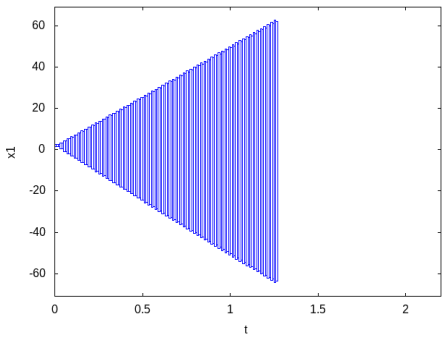
(c) QR preconditioning



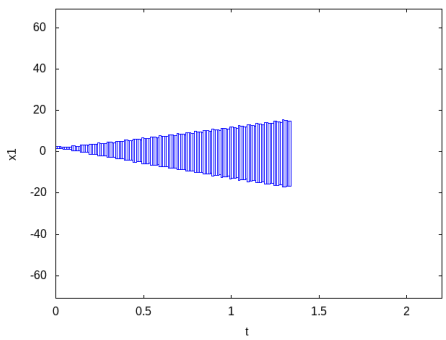
(d) No processing



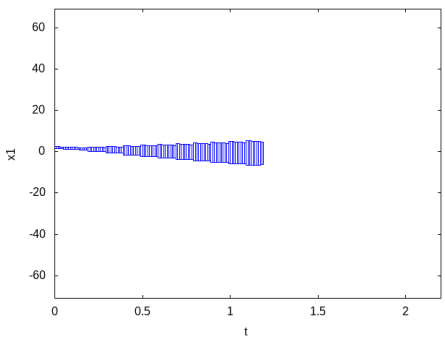
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

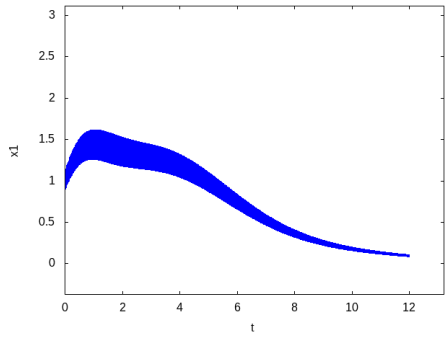


(g) Shrink wrapping after 5 time steps

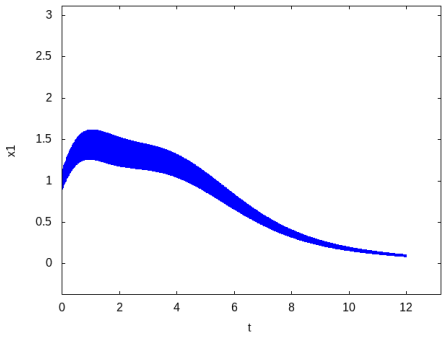


(h) Shrink wrapping after 10 time steps

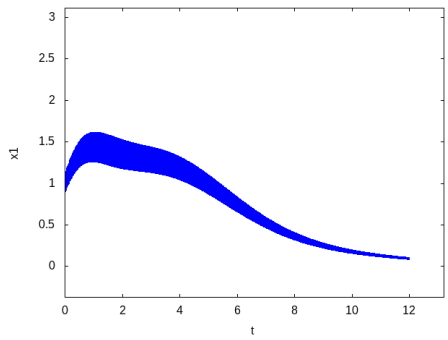
Figure B.24: Flowpipes for Two tanks system using different processing methods.



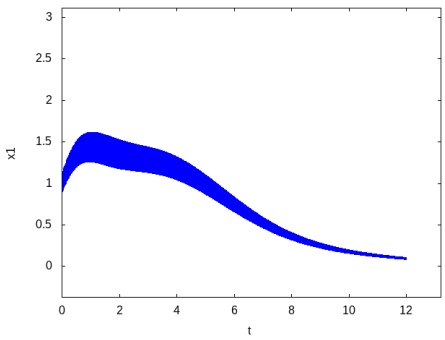
(a) Identity preconditioning



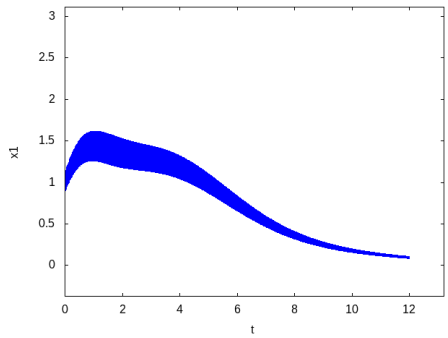
(b) Parallelepiped preconditioning



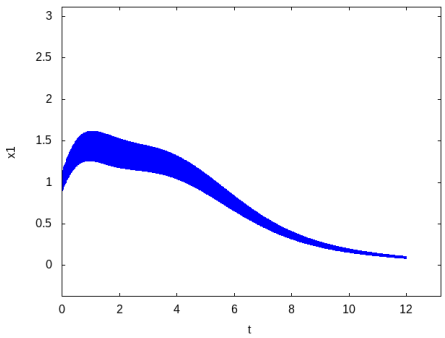
(c) QR preconditioning



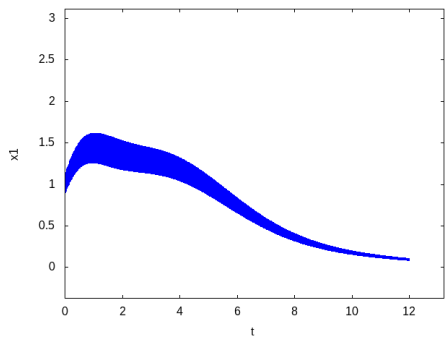
(d) No processing



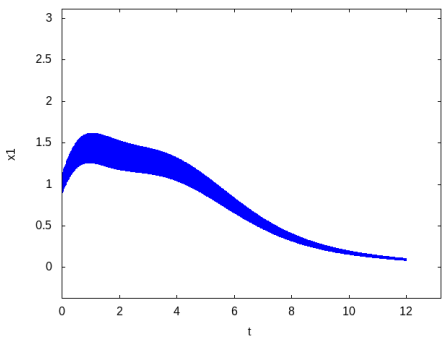
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

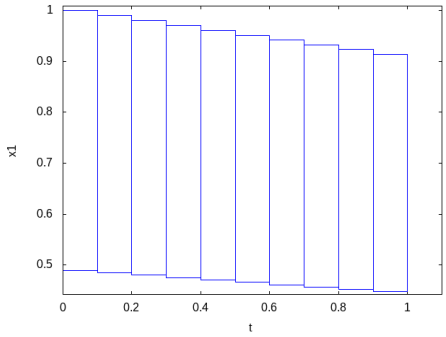


(g) Shrink wrapping after 5 time steps

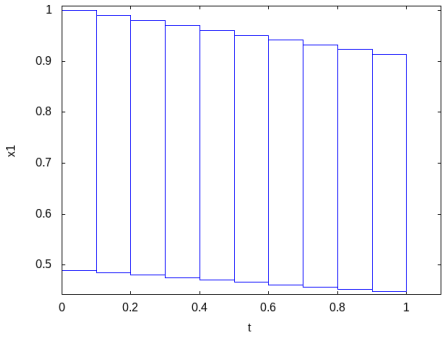


(h) Shrink wrapping after 10 time steps

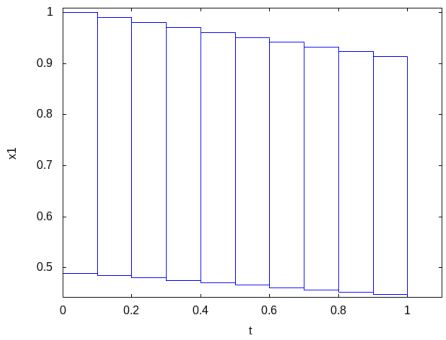
Figure B.25: Flowpipes for Three vehicle system using different processing methods.



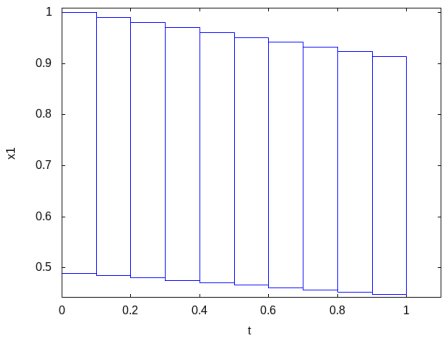
(a) Identity preconditioning



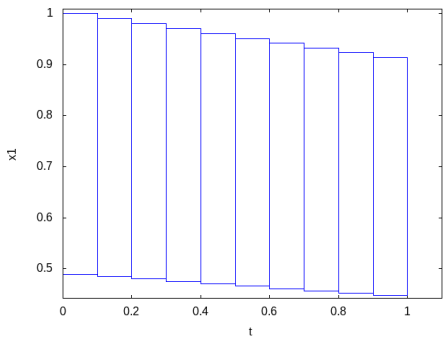
(b) Parallelepiped preconditioning



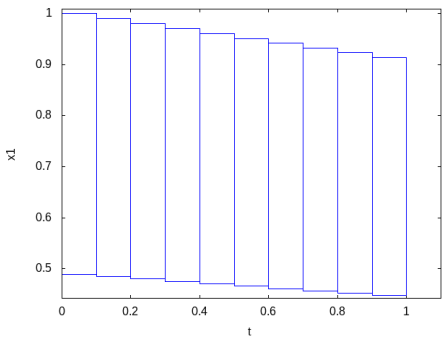
(c) QR preconditioning



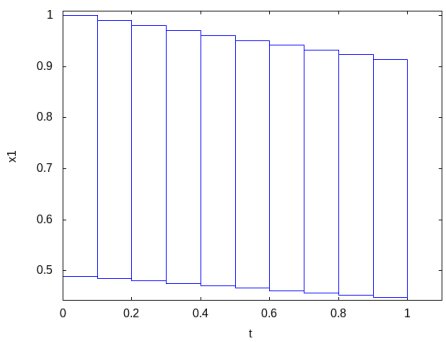
(d) No processing



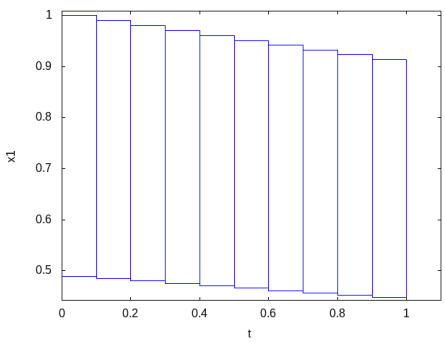
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

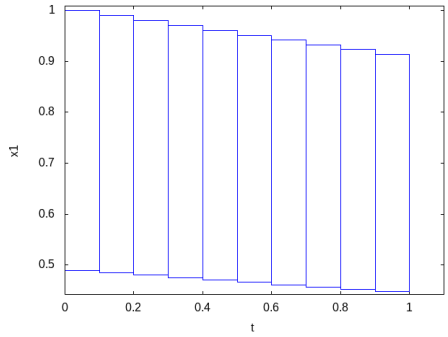


(g) Shrink wrapping after 5 time steps

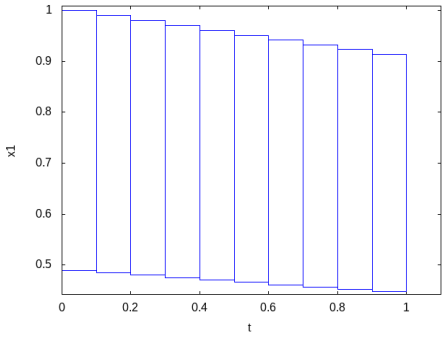


(h) Shrink wrapping after 10 time steps

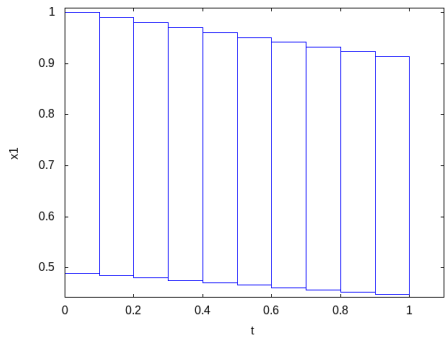
Figure B.26: Flowpipes for Linear system using different processing methods.



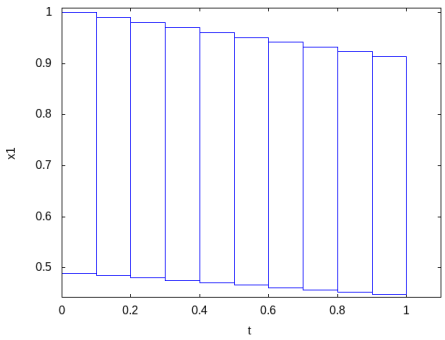
(a) Identity preconditioning



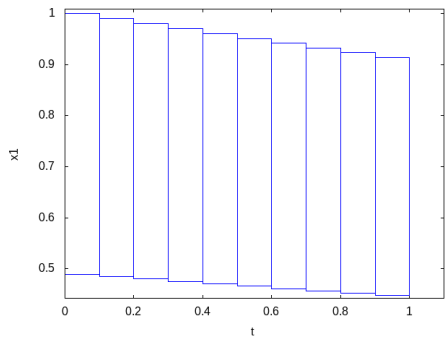
(b) Parallelepiped preconditioning



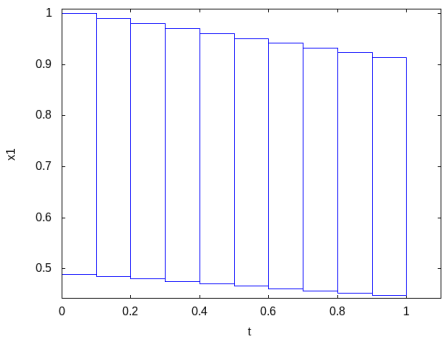
(c) QR preconditioning



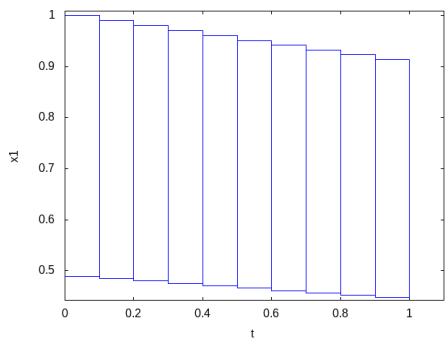
(d) No processing



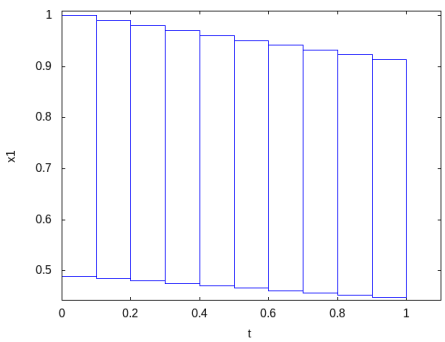
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

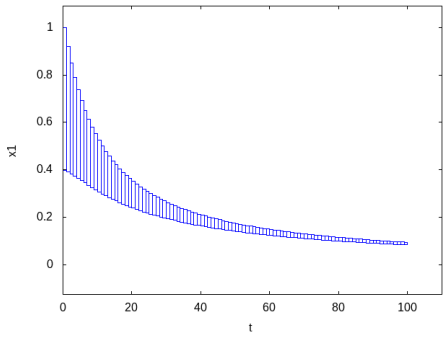


(g) Shrink wrapping after 5 time steps

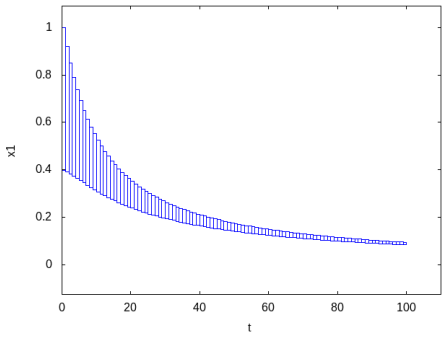


(h) Shrink wrapping after 10 time steps

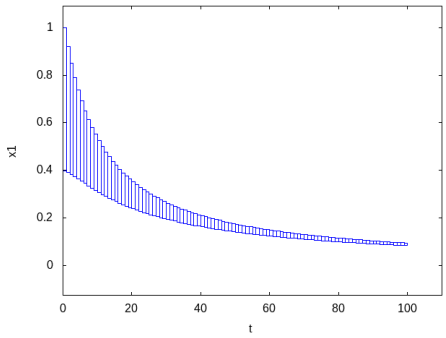
Figure B.27: Flowpipes for Lin-dep system using different processing methods.



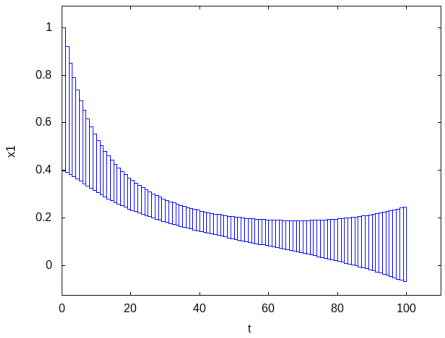
(a) Identity preconditioning



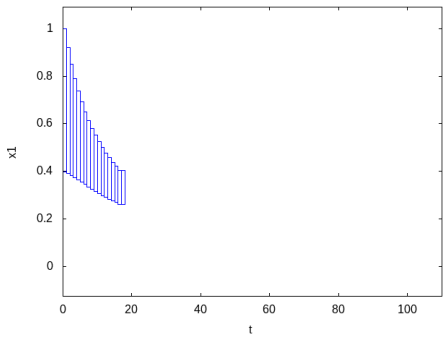
(b) Parallelepiped preconditioning



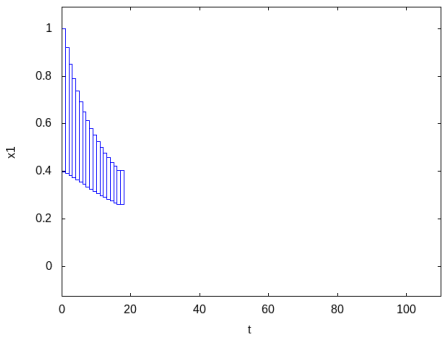
(c) QR preconditioning



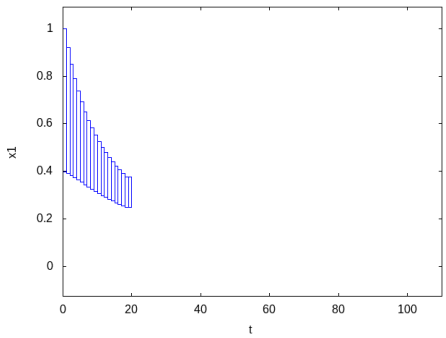
(d) No processing



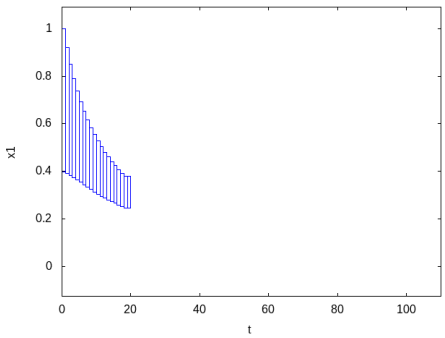
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps

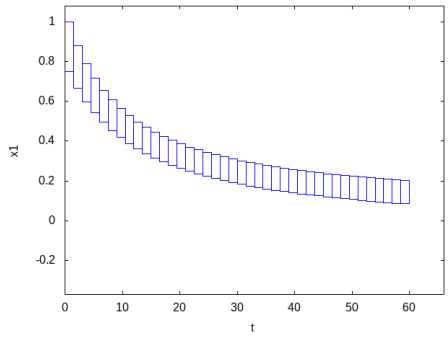


(g) Shrink wrapping after 5 time steps

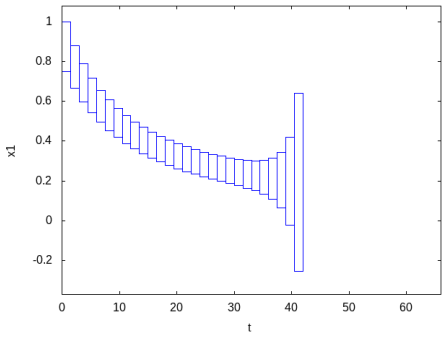


(h) Shrink wrapping after 10 time steps

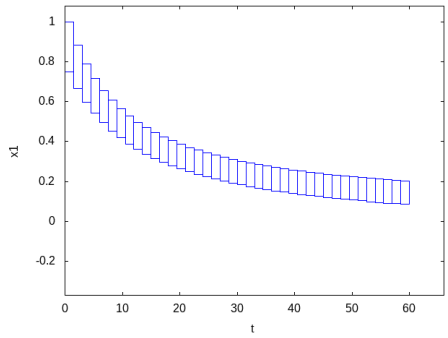
Figure B.28: Flowpipes for Sqr-deg system using different processing methods.



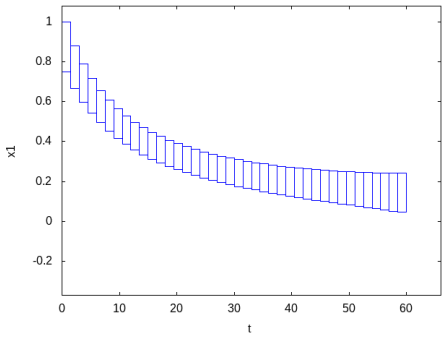
(a) Identity preconditioning



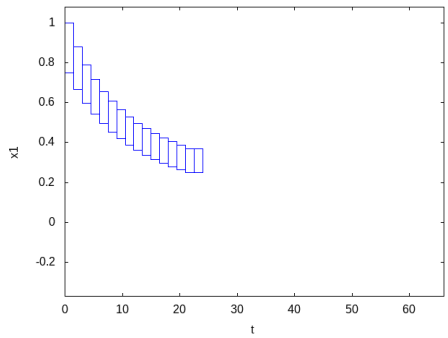
(b) Parallelepiped preconditioning



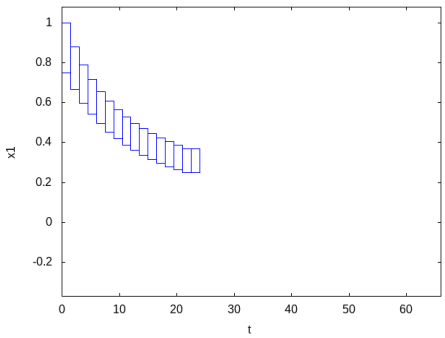
(c) QR preconditioning



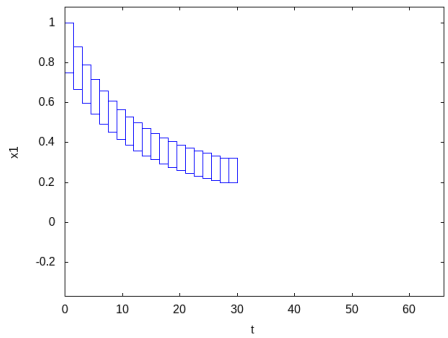
(d) No processing



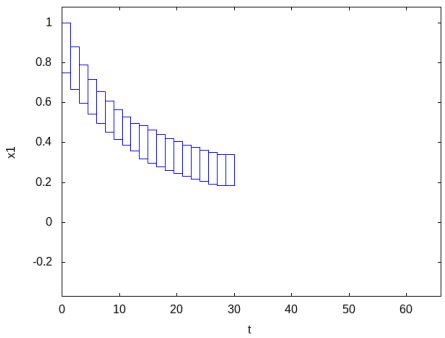
(e) Shrink wrapping at every time step



(f) Shrink wrapping after 2 time steps



(g) Shrink wrapping after 5 time steps



(h) Shrink wrapping after 10 time steps

Figure B.29: Flowpipes for Pairwise system using different processing methods.

Method	Integration progress	Width
Identity preconditioning	1000.000	0.00676471
Parallelepiped preconditioning	0.000	–
QR preconditioning	1000.000	0.0068771
No processing	570.000	MAX
Shrink wrapping 1	10.000	0.02
Shrink wrapping 2	40.000	0.0987537
Shrink wrapping 5	50.000	0.0599736
Shrink wrapping 10	100.000	0.0537586

Table B.1: Processing experiments for AND-Gate.

Method	Integration progress	Width
Identity preconditioning	40.000	0.0243657
Parallelepiped preconditioning	14.550	0.0324645
QR preconditioning	31.750	41.7771
No processing	40.000	0.117947
Shrink wrapping 1	0.600	MAX
Shrink wrapping 2	0.800	MAX
Shrink wrapping 5	1.250	MAX
Shrink wrapping 10	1.850	44.1445

Table B.2: Processing experiments for AND-OR Gate.

We will present the time the integration method managed to integrate to and the the width of the interval bounding the first variable of the system. The width can be a real number, MAX if the method broke down due to not being able to compute valid interval remainder (an effect of it being larger than what our tool supports) or nothing if the method was not applicable at all.

The data used in Section 7.3 is presented in Tables B.1 to B.29.

Method	Integration progress	Width
Identity preconditioning	6.000	MAX
Parallelepiped preconditioning	2.400	MAX
QR preconditioning	15.000	0.130658
No processing	1.890	MAX
Shrink wrapping 1	1.380	0.0723647
Shrink wrapping 2	1.380	0.0729955
Shrink wrapping 5	1.350	0.075612
Shrink wrapping 10	1.500	0.084173

Table B.3: Processing experiments for Brusselator.

Method	Integration progress	Width
Identity preconditioning	7.150	MAX
Parallelepiped preconditioning	4.820	MAX
QR preconditioning	10.000	0.873236
No processing	10.000	2.29083
Shrink wrapping 1	0.900	0.231232
Shrink wrapping 2	0.900	0.230867
Shrink wrapping 5	0.900	0.230644
Shrink wrapping 10	0.900	0.230565

Table B.4: Processing experiments for Buckling col.

Method	Integration progress	Width
Identity preconditioning	2.910	MAX
Parallelepiped preconditioning	10.020	0.030186
QR preconditioning	8.730	MAX
No processing	1.650	MAX
Shrink wrapping 1	4.350	0.166456
Shrink wrapping 2	4.320	0.18811
Shrink wrapping 5	4.200	0.27825
Shrink wrapping 10	2.700	0.717896

Table B.5: Processing experiments for Jet engine.

Method	Integration progress	Width
Identity preconditioning	1.725	MAX
Parallelepiped preconditioning	0.627	MAX
QR preconditioning	5.397	MAX
No processing	0.912	MAX
Shrink wrapping 1	0.579	0.0636179
Shrink wrapping 2	0.582	0.0658929
Shrink wrapping 5	0.585	0.0672701
Shrink wrapping 10	0.600	0.0796562

Table B.6: Processing experiments for Lorentz.

Method	Integration progress	Width
Identity preconditioning	3.300	MAX
Parallelepiped preconditioning	10.000	2.03082
QR preconditioning	6.500	MAX
No processing	2.140	MAX
Shrink wrapping 1	5.340	0.744197
Shrink wrapping 2	5.360	0.735598
Shrink wrapping 5	5.400	0.794035
Shrink wrapping 10	5.400	0.939039

Table B.7: Processing experiments for Lotka-Volterra.

Method	Integration progress	Width
Identity preconditioning	10.000	2.53292
Parallelepiped preconditioning	10.000	2.44783
QR preconditioning	10.000	2.44783
No processing	10.000	2.79329
Shrink wrapping 1	10.000	2.44792
Shrink wrapping 2	10.000	2.44792
Shrink wrapping 5	10.000	2.44795
Shrink wrapping 10	10.000	2.44806

Table B.8: Processing experiments for Moore rot.

Method	Integration progress	Width
Identity preconditioning	6.000	1.34766
Parallelepiped preconditioning	0.000	–
QR preconditioning	6.000	1.31313
No processing	6.000	4.88364
Shrink wrapping 1	0.020	0.0
Shrink wrapping 2	0.040	0.176583
Shrink wrapping 5	0.040	0.192306
Shrink wrapping 10	0.200	0.284631

Table B.9: Processing experiments for Roessler.

Method	Integration progress	Width
Identity preconditioning	7.000	0.555849
Parallelepiped preconditioning	2.860	MAX
QR preconditioning	7.000	0.176818
No processing	7.000	0.234957
Shrink wrapping 1	0.320	0.281722
Shrink wrapping 2	0.320	0.281378
Shrink wrapping 5	0.400	0.249124
Shrink wrapping 10	0.400	0.248954

Table B.10: Processing experiments for Vanderpol.

Method	Integration progress	Width
Identity preconditioning	10.000	9.96095
Parallelepiped preconditioning	0.000	–
QR preconditioning	10.000	9.96095
No processing	10.000	9.96095
Shrink wrapping 1	0.100	0.2
Shrink wrapping 2	0.200	0.24905
Shrink wrapping 5	1.000	1.03385
Shrink wrapping 10	2.000	2.01485

Table B.11: Processing experiments for Bouncing ball.

Method	Integration progress	Width
Identity preconditioning	100.100	285.505
Parallelepiped preconditioning	100.100	285.505
QR preconditioning	100.100	285.505
No processing	100.100	285.505
Shrink wrapping 1	100.100	285.505
Shrink wrapping 2	100.100	285.505
Shrink wrapping 5	100.100	285.505
Shrink wrapping 10	100.100	285.505

Table B.12: Processing experiments for Cruise control.

Method	Integration progress	Width
Identity preconditioning	360.000	0.623517
Parallelepiped preconditioning	0.000	–
QR preconditioning	360.000	0.626963
No processing	2.900	MAX
Shrink wrapping 1	0.100	4.0
Shrink wrapping 2	0.200	4.00409
Shrink wrapping 5	1.800	MAX
Shrink wrapping 10	2.400	MAX

Table B.13: Processing experiments for Glycemic 1.

Method	Integration progress	Width
Identity preconditioning	360.000	0.201601
Parallelepiped preconditioning	0.000	–
QR preconditioning	360.000	1.68426
No processing	2.900	MAX
Shrink wrapping 1	0.100	4.0
Shrink wrapping 2	0.200	4.00409
Shrink wrapping 5	1.800	MAX
Shrink wrapping 10	2.400	MAX

Table B.14: Processing experiments for Glycemic 2.

Method	Integration progress	Width
Identity preconditioning	4.000	5.56115e-05
Parallelepiped preconditioning	0.000	–
QR preconditioning	4.000	5.56115e-05
No processing	4.000	5.56167e-05
Shrink wrapping 1	0.050	0.1
Shrink wrapping 2	2.800	MAX
Shrink wrapping 5	3.750	MAX
Shrink wrapping 10	4.000	MAX

Table B.15: Processing experiments for Filtered osc 4.

Method	Integration progress	Width
Identity preconditioning	4.000	5.56115e-05
Parallelepiped preconditioning	0.000	–
QR preconditioning	4.000	5.56116e-05
No processing	4.000	5.56167e-05
Shrink wrapping 1	0.050	0.1
Shrink wrapping 2	1.600	MAX
Shrink wrapping 5	2.500	MAX
Shrink wrapping 10	2.500	MAX

Table B.16: Processing experiments for Filtered osc 8.

Method	Integration progress	Width
Identity preconditioning	4.000	5.56115e-05
Parallelepiped preconditioning	0.000	–
QR preconditioning	4.000	5.56117e-05
No processing	4.000	5.56167e-05
Shrink wrapping 1	0.050	0.1
Shrink wrapping 2	0.900	MAX
Shrink wrapping 5	1.500	MAX
Shrink wrapping 10	2.000	MAX

Table B.17: Processing experiments for Filtered osc 16.

Method	Integration progress	Width
Identity preconditioning	4.000	5.56115e-05
Parallelepiped preconditioning	0.000	–
QR preconditioning	4.000	5.56118e-05
No processing	4.000	5.56167e-05
Shrink wrapping 1	0.050	0.1
Shrink wrapping 2	0.200	0.122817
Shrink wrapping 5	1.250	MAX
Shrink wrapping 10	1.500	1.88874

Table B.18: Processing experiments for Filtered osc 32.

Method	Integration progress	Width
Identity preconditioning	98.780	MAX
Parallelepiped preconditioning	78.680	MAX
QR preconditioning	98.780	MAX
No processing	23.600	MAX
Shrink wrapping 1	51.920	0.65673
Shrink wrapping 2	51.920	0.657626
Shrink wrapping 5	52.000	0.659381
Shrink wrapping 10	52.000	0.659723

Table B.19: Processing experiments for Neuron 1.

Method	Integration progress	Width
Identity preconditioning	10.240	MAX
Parallelepiped preconditioning	10.200	MAX
QR preconditioning	10.240	MAX
No processing	10.220	MAX
Shrink wrapping 1	8.260	9.92666
Shrink wrapping 2	8.280	10.1464
Shrink wrapping 5	8.300	10.3658
Shrink wrapping 10	8.400	11.4099

Table B.20: Processing experiments for Neuron 2.

Method	Integration progress	Width
Identity preconditioning	7.500	0.01
Parallelepiped preconditioning	0.000	–
QR preconditioning	7.500	0.01
No processing	7.500	0.01
Shrink wrapping 1	0.000	–
Shrink wrapping 2	0.020	0.01
Shrink wrapping 5	0.100	0.01
Shrink wrapping 10	0.200	0.01

Table B.21: Processing experiments for Non-holonomic.

Method	Integration progress	Width
Identity preconditioning	50.000	MAX
Parallelepiped preconditioning	0.000	–
QR preconditioning	50.000	MAX
No processing	50.000	MAX
Shrink wrapping 1	0.100	10.0
Shrink wrapping 2	0.200	10.201
Shrink wrapping 5	16.600	MAX
Shrink wrapping 10	27.000	MAX

Table B.22: Processing experiments for Rod reactor.

Method	Integration progress	Width
Identity preconditioning	0.100	0.255531
Parallelepiped preconditioning	0.000	–
QR preconditioning	0.100	0.286625
No processing	0.100	0.279511
Shrink wrapping 1	0.010	0.0
Shrink wrapping 2	0.100	0.538585
Shrink wrapping 5	0.100	0.297048
Shrink wrapping 10	0.100	0.270653

Table B.23: Processing experiments for Switching.

Method	Integration progress	Width
Identity preconditioning	2.000	0.160753
Parallelepiped preconditioning	0.000	–
QR preconditioning	2.000	0.177613
No processing	2.000	0.272389
Shrink wrapping 1	0.010	1.0
Shrink wrapping 2	2.000	201.726
Shrink wrapping 5	2.000	49.9939
Shrink wrapping 10	2.000	21.8875

Table B.24: Processing experiments for Two tanks.

Method	Integration progress	Width
Identity preconditioning	12.000	0.0233121
Parallelepiped preconditioning	12.000	0.0233117
QR preconditioning	12.000	0.0233116
No processing	12.000	0.0233116
Shrink wrapping 1	12.000	0.0233116
Shrink wrapping 2	12.000	0.0233116
Shrink wrapping 5	12.000	0.0233116
Shrink wrapping 10	12.000	0.0233116

Table B.25: Processing experiments for Three vehicle.

Method	Integration progress	Width
Identity preconditioning	1.000	0.466105
Parallelepiped preconditioning	1.000	0.466105
QR preconditioning	1.000	0.466105
No processing	1.000	0.466105
Shrink wrapping 1	1.000	0.466105
Shrink wrapping 2	1.000	0.466105
Shrink wrapping 5	1.000	0.466105
Shrink wrapping 10	1.000	0.466105

Table B.26: Processing experiments for Linear.

Method	Integration progress	Width
Identity preconditioning	1.000	0.466105
Parallelepiped preconditioning	1.000	0.466105
QR preconditioning	1.000	0.466105
No processing	1.000	0.466105
Shrink wrapping 1	1.000	0.466105
Shrink wrapping 2	1.000	0.466105
Shrink wrapping 5	1.000	0.466105
Shrink wrapping 10	1.000	0.466105

Table B.27: Processing experiments for Lin-dep.

Method	Integration progress	Width
Identity preconditioning	100.000	0.0109325
Parallelepiped preconditioning	100.000	0.0109325
QR preconditioning	100.000	0.0109325
No processing	100.000	0.314886
Shrink wrapping 1	18.000	0.143177
Shrink wrapping 2	18.000	0.143293
Shrink wrapping 5	20.000	0.127603
Shrink wrapping 10	20.000	0.131215

Table B.28: Processing experiments for Sqr-deg.

Method	Integration progress	Width
Identity preconditioning	60.000	0.117493
Parallelepiped preconditioning	45.000	MAX
QR preconditioning	60.000	0.116735
No processing	60.000	0.195796
Shrink wrapping 1	24.000	0.122651
Shrink wrapping 2	24.000	0.122975
Shrink wrapping 5	30.000	0.123299
Shrink wrapping 10	30.000	0.156504

Table B.29: Processing experiments for Pairwise.

Bibliography

[ABDM00] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, pages 20–31, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[ABHS07] Wolfgang Ahrendt, Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt. KeY: A formal method for object-oriented systems. In Marcello M. Bonsangue and Einar Broch Johnsen, editors, *Formal Methods for Open Object-Based Distributed Systems*, pages 32–43, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[ADG03] Eugene Asarin, Thao Dang, and Antoine Girard. Reachability analysis of nonlinear systems using conservative approximation. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, pages 20–35, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[ADG07] Eugene Asarin, Thao Dang, and Antoine Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, Feb 2007.

[Apo67] T.M. Apostol. *Calculus: One-variable calculus, with an introduction to linear algebra*. Calculus. Wiley, 1967.

[Apo69] T.M. Apostol. *Calculus: Multi-variable calculus and linear algebra, with applications to differential equations and probability*. Calculus. Wiley, 1969.

[APS06] Erin M. Aylward, Pablo A. Parrilo, and Jean-Jacques E. Slotine. Stability and Robustness Analysis of Nonlinear Systems via Contraction

Metrics and SOS Programming. *arXiv Mathematics e-prints*, page math/0603313, Mar 2006.

[ASB08] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *2008 47th IEEE Conference on Decision and Control*, pages 4042–4048, Dec 2008.

[BCD13] Olivier Bouissou, Alexandre Chapoutot, and Adel Djoudi. Enclosing temporal evolution of dynamical systems using numerical methods. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Methods*, pages 108–123, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BM98] Martin Berz and Kyoko Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4):361–369, 1998.

[Car13] Luca Cardelli. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science*, 23(2):247–271, 2013.

[CÁS12] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 183–192. IEEE, 2012.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 258–263, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[CDS+13] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srnivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, September 2013.

[Che15] Xin Chen. *Computing rigorous bounds on the solution of an initial value problem for an ordinary differential equation*. PhD thesis, RWTH Aachen University, 2015.

[CK03] A. Chutinan and B. H. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, Jan 2003.

[CK04] Martine Ceberio and Vladik Kreinovich. Greedy algorithms for optimizing multivariate Horner schemes. *SIGSAM Bull.*, 38(1):8–15, March 2004.

[CR96] George F Corliss and Robert Rihm. Validating an a priori enclosure using high-order Taylor series. *MATHEMATICAL RESEARCH*, 90:228–238, 1996.

[CS16] X. Chen and S. Sankaranarayanan. Decomposed reachability analysis for nonlinear systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 13–24, Nov 2016.

[DLGM09] Thao Dang, Colas Le Guernic, and Oded Maler. Computing reachable states for nonlinear biological models. In Pierpaolo Degano and Roberto Gorrieri, editors, *Computational Methods in Systems Biology*, pages 126–141, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DM07] Alexandre Donzé and Oded Maler. Systematic simulation using sensitivity analysis. In *International Workshop on Hybrid Systems: Computation and Control*, pages 174–189. Springer, 2007.

[DMT10] Thao Dang, Oded Maler, and Romain Testylier. Accurate hybridization of nonlinear systems. In *Proc. of Intl. Conf. on Hybrid Systems: Computation and Control*, HSCC '10, pages 11–20, New York, NY, USA, 2010. ACM.

[Don07] Alexandre Donzé. *Trajectory-based Verification and Controller Synthesis for Continuous and Hybrid Systems*. PhD thesis, University Joseph Fouriee, 2007.

[DP04] Robert M Dirks and Niles A Pierce. Triggered amplification by hybridization chain reaction. *Proceedings of the National Academy of Sciences*, 101(43):15275–15278, 2004.

[DRJ13] Yi Deng, Akshay Rajhans, and A. Agung Julius. STRONG: A Trajectory-Based Verification Toolbox for Hybrid Systems. In Kaustubh

Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio, editors, *Quantitative Evaluation of Systems*, pages 165–168, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Eij81] Pieter Eijgenraam. The solution of initial value problems using interval arithmetic: Formulation and analysis of an algorithm. *MC Tracts*, 144:1–185, 1981.

[ERNF11] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *Proceedings of the 9th International Conference on Software Engineering and Formal Methods*, SEFM'11, pages 172–187, Berlin, Heidelberg, 2011. Springer-Verlag.

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1:209–236, 01 2007.

[Fis91] Michael E Fisher. A semiclosed-loop algorithm for the control of blood glucose levels in diabetics. *IEEE transactions on biomedical engineering*, 38(1):57–61, 1991.

[FKJM16] Chuchu Fan, James Kapinski, Xiaoqing Jin, and Sayan Mitra. Locally optimal reach set over-approximation for nonlinear systems. In *Proceedings of the 13th International Conference on Embedded Software*, EMSOFT '16, pages 6:1–6:10, New York, NY, USA, 2016. ACM.

[FKS+85] Stuart M Furler, Edward W Kraegen, Robert H Smallwood, Donald J Chisholm, et al. Blood glucose control by intermittent loop closure in the basal mode: computer simulation studies with a diabetic model. *Diabetes care*, 8(6):553–561, 1985.

[FLGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395. Springer, 2011.

[FQM+16] Chuchu Fan, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, and Parasara Sridhar Duggirala. Automatic Reachability Analysis for Nonlinear Hybrid Models with C2E2. In Swarat Chaudhuri and Azadeh

Farzan, editors, *Computer Aided Verification*, pages 531–538, Cham, 2016. Springer International Publishing.

[Gir05] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 291–305, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[GKC13] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT Solver for Nonlinear Theories over the Reals. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, pages 208–214, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[GKM82] Susan L Graham, Peter B Kessler, and Marshall K Mckusick. Gprof: A call graph execution profiler. In *ACM Sigplan Notices*, volume 17, pages 120–126. ACM, 1982.

[GLGM06] Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In João P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control*, pages 257–271, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[GP06] Antoine Girard and George J. Pappas. Verification using simulation. In João P. Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control*, pages 272–286, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Hes11] Henry Hess. Engineering applications of biomolecular motors. *Annual review of biomedical engineering*, 13:429–450, 2011.

[HJVE01] T. Hickey, Q. Ju, and M. H. Van Emden. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068, September 2001.

[HN10] Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In Matthias Blume, Naoki Kobayashi, and Germán Vidal, editors, *Functional and Logic Programming*, pages 103–117, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[HyP19] HyPro. HyPro benchmarks. https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/, 2019. [Online; accessed 10-June-2019].

[Imm14] Fabian Immler. Formally verified computation of enclosures of solutions of ordinary differential equations. In *NASA Formal Methods*, pages 113–127. Springer, 2014.

[JN02] Kenneth R. Jackson and Nedialko S. Nedialkov. Some recent advances in validated methods for IVPs for ODEs. *Applied Numerical Mathematics*, 42(1):269 – 284, 2002. Numerical Solution of Differential and Differential-Algebraic Equations, 4-9 September 2000, Halle, Germany.

[KGBM04] Michal Kvasnica, Pascal Grieder, Mato Baotić, and Manfred Morari. Multi-parametric toolbox (MPT). In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, pages 448–462, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[KGCC15] S. Kong, S. Gao, W. Chen, and E. Clarke. dReach: δ-reachability analysis for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015.

[KRO05] Otilia M Koo, Israel Rubinstein, and Hayat Onyuksel. Role of nanotechnology in targeted drug delivery and imaging: a concise review. *Nanomedicine: Nanotechnology, Biology and Medicine*, 1(3):193–212, 2005.

[KV07] A. A. Kurzhanskiy and P. Varaiya. Ellipsoidal techniques for reachability analysis of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 52(1):26–38, Jan 2007.

[Loh87] Rudolph J Lohner. Enclosing the solutions of ordinary initial and boundary value problems. *Computer Arithmetic, Scientific Computation and Programming Languages*, pages 255–286, 1987.

[Loh88] R. Lohner. *Einschliessung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. Universität Karlsruhe, 1988.

[Loh95] R.J. Lohner. Step size order control in the verified solution of IVP with ODEs. In *SciCADE 95 International Conference on Scientific Computation and Differential Equations*, 1995.

[LPC+12] Matthew R Lakin, David Parker, Luca Cardelli, Marta Kwiatkowska, and Andrew Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 9(72):1470–1485, 2012.

[LS07] Youdong Lin and Mark A. Stadtherr. Validated solutions of initial value problems for parametric ODEs. *Appl. Numer. Math.*, 57(10):1145–1162, October 2007.

[LYCP11] Matthew R Lakin, Simon Youssef, Luca Cardelli, and Andrew Phillips. Abstractions for DNA circuit design. *Journal of The Royal Society Interface*, 9(68):470–486, 2011.

[Mak98] K. Makino. *Rigorous Analysis of Nonlinear Motion in Particle Accelerators*. Michigan State University. Department of Physics and Astronomy, 1998.

[MB11] Kyoko Makino and Martin Berz. Suppression of the wrapping effect by Taylor model-based verified integrators: Long-term stabilization by preconditioning. *International Journal of Differential Equations and Applications*, 10(4), 2011.

[MBT11] Richard A Muscat, Jonathan Bath, and Andrew J Turberfield. A programmable molecular robot. *Nano letters*, 11(3):982–987, 2011.

[Mei07] J.D. Meiss. *Differential Dynamical Systems*. Mathematical Modeling and Computation. SIAM, 2007.

[Mic15a] Microsoft Research. A programming language for DNA computing. `http://research.microsoft.com/en-us/projects/dna/`, 2015. [Online; accessed 31-July-2015].

[Mic15b] Microsoft Research. Visual DSD. `http://dsd.azurewebsites.net/`, 2015. [Online; accessed 31-July-2015].

[MKC09] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[Moo65] Ramon E Moore. Automatic local coordinate transformations to reduce the growth of error bounds in interval computation of solutions of ordinary differential equations. *Error in digital computation*, 2:103–140, 1965.

[Moo66] R.E. Moore. *Interval analysis*. Prentice-Hall series in automatic computation. Prentice-Hall, 1966.

[Ned11] N. S. Nedialkov. *Implementing a Rigorous ODE Solver Through Literate Programming*, pages 3–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[NJN07] M. Neher, K. R. Jackson, and N. S. Nedialkov. On Taylor model based integration of ODEs. *SIAM J. Numer. Anal*, 45, 2007.

[NJP01] Nedialko S Nedialkov, Kenneth R Jackson, and John D Pryce. An effective high-order interval method for validating existence and uniqueness of the solution of an IVP for an ODE. *Reliable Computing*, 7(6):449–465, 2001.

[Oeh11] Jens Oehlerking. *Decomposition of stability proofs for hybrid systems*. PhD thesis, Universität Oldenburg, 2011.

[OSS09] Tosan Omabegho, Ruojie Sha, and Nadrian C Seeman. A bipedal DNA Brownian motor with coordinated legs. *Science*, 324(5923):67–71, 2009.

[Pau93] Lawrence C. Paulson. Isabelle: The next 700 theorem provers. *CoRR*, cs.LO/9301106, 1993.

[PC09] Andrew Phillips and Luca Cardelli. A programming language for composable DNA circuits. *Journal of the Royal Society Interface*, 6(suppl_4):S419–S436, 2009.

[Pla08] André Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, Aug 2008.

[PQ08] A. Platzer and J. Quesel. KeYmaera: A hybrid theorem prover for hybrid systems (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*. Springer, 2008.

[PV11] P. Prabhakar and M. Viswanathan. A dynamic algorithm for approximate flow computations. In *Proc. Intl Conf. on Hybrid Systems: Computation and Control*, HSCC '11. ACM, 2011.

[PW09] Priscilla EM Purnick and Ron Weiss. The second wave of synthetic biology: from modules to systems. *Nature reviews Molecular cell biology*, 10(6):410, 2009.

[QW11] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[QWB11] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368, 2011.

[Rot06] Paul WK Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297, 2006.

[See10] Nadrian C Seeman. Nanomaterials based on DNA. *Annual review of biochemistry*, 79:65–87, 2010.

[SSW10] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

[SSZW06] Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-free nucleic acid logic circuits. *science*, 314(5805):1585–1588, 2006.

[TD13] R. Testylier and T. Dang. NLTOOLBOX: A library for reachability computation of nonlinear dynamical systems. In D. Van Hung and M. Ogawa, editors, *Automated Technology for Verification and Analysis*. Springer, 2013.

[VHMK97] Pascal Van Hentenryck, David McAllester, and Deepak Kapur. Solving polynomial systems using a branch and prune approach. *SIAM J. Numer. Anal.*, 34(2):797–827, April 1997.

[VVS99] Frits W Vaandrager and Jan H Van Schuppen. *Hybrid Systems: Computation and Control: Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29-31, 1999 Proceedings*. Springer Science & Business Media, 1999.

[Wan77] Gerhard Wanner. *On the integration of stiff differential equations*, pages 209–226. Birkhäuser Basel, Basel, 1977.

[WLWS98] Erik Winfree, Furong Liu, Lisa A Wenzler, and Nadrian C Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539, 1998.

[YWH+13] Boyan Yordanov, Christoph Wintersteiger, Youssef Hamadi, Andrew Phillips, and Hillel Kugler. Functional analysis of large-scale DNA strand displacement circuits. *International Conference on DNA Computing and Molecular Programming*, 8141:189–203, September 2013.

[YWHK13] Boyan Yordanov, Christoph M. Wintersteiger, Youssef Hamadi, and Hillel Kugler. *SMT-Based Analysis of Biological Computation*, pages 78–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Zha92] Feng Zhao. Automatic analysis and synthesis of controllers for dynamical systems based on phase-space knowledge. Technical report, Cambridge, MA, USA, 1992.

[ZRK11] Fuzhong Zhang, Sarah Rodriguez, and Jay D Keasling. Metabolic engineering of microbial pathways for advanced biofuels production. *Current opinion in biotechnology*, 22(6):775–783, 2011.

[ZS11] David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103, 2011.

Nomenclature

Mag Magnitude of an interval, page 16

Mid Midpoint of an interval, page 16

W Width of an interval, page 16

$\mathbb{IR}$ The set of all real intervals, page 16

$\mathbf{i}$ Interval, page 15

ip Function returning the initial condition parameter for a variable, page 28

pv Function associating variables with left model parameters, page 28

$rem(\vec{\mathcal{T}})$ Vector of remainder intervals of $\vec{\mathcal{T}}$, page 27

$\mathcal{T}$ Taylor model, page 23

$\mathcal{T}.poly$ Polynomial of the Taylor model, page 23

$\mathcal{T}.rem$ Remainder of the Taylor model, page 23

$\mathcal{T}_x$ Element of $\vec{\mathcal{T}}$ corresponding to variable x, page 23

C Matrix of coefficients of the desired linear part in preconditioning, page 46

Δ Integration period, page 34

$\vec{\phi}(t;\vec{x}_{init})$ Solution to IVP, page 31

$\vec{\phi}(\vec{X}_{init})$ Family of functions of the solutions for set initial conditions $\vec{X}_{init}$, page 32

$\vec{\Phi}(\vec{X}_{init})(t)$ Solution to set initial condition $\vec{X}_{init}$, page 32

$\vec{x}_{init}$ Point initial condition, page 31